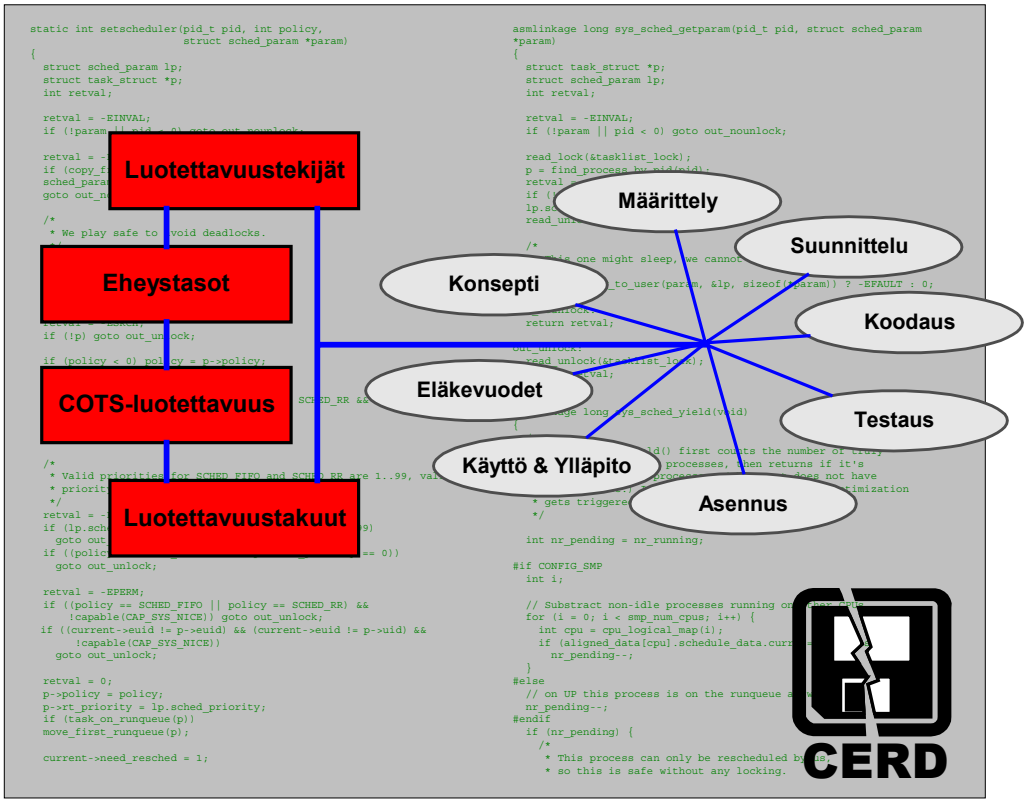


Hannu Harju

Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi

Osa 1

VTT TIEDOTTEITA – MEDDELANDEN – RESEARCH NOTES 2151

Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi

Osa 1

Hannu Harju

VTT Tuotteet ja tuotanto

ISBN 951–38–6062–0 (nid.)
ISSN 1235–0605 (nid.)

ISBN 951–38–6063–9 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1235–0605 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Tuotteet ja tuotanto, Tekniikantie 12, PL 1301, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 6752

VTT Industriella system, Teknikvägen 12, PB 1301, 02044 VTT
tel. växel (09) 4561, fax (09) 456 6752

VTT Industrial Systems, Tekniikantie 12, P.O.Box 1301, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 6752

Otamedia Oy, Espoo 2002

Harju, Hannu. Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi. Osa 1 [Cost-effective design and assessment of dependable software. Part 1]. Espoo 2002. VTT Tiedotteita – Research Notes 2151. 114 s. + liitt. 15 s.

Avainsanat software dependability, safety integrity levels, reliability scoring, software reliability engineering, risk management process

Tiivistelmä

Ohjelmistojen käyttäminen kriittisiin sovelluksiin on jatkuvassa kasvussa. Päinvastoin kuin laitteistoviat, ohjelmistoviat ovat systemaattisia ja ohjelmistovirheet voivat piileksiä pitkiä aikoja ennen paljastumistaan. Tässä tiedotteessa tarkastellaan ensiksi yleisesti eräitä ohjelmiston luotettavuuteen vaikuttavia tärkeimpiä tekijöitä, sitten erityisteemoina eheystason hyödyntämistä, valmisohjelmistojen luotettavuutta ja ohjelmiston luotettavuustakuita.

Monet turvallisuuskriittisten ohjelmistojen kehitysstandardit tukeutuvat käsitteeseen turvallisuuden eheystaso. Se on todennäköisyysmitta sille, että turvatoiminto täyttää sille asetetut turvallisuusvaatimukset. Turvallisuuden eheystasomenettelyllä on ilmeiset etunsa mutta myös ongelmansa, jotka vaikeuttavat menettelyn hyödyntämistä käytännössä. Tiedote sisältää lyhyen viitestandardien tarkastelun tavoitteena selvittää lähestymistavan hyödyntämismahdollisuudet muille luotettavuusattribuuteille kuin turvallisuus.

Kaupallisesti saatavilla oleva ohjelmistokomponentti, jota usein kutsutaan COTSiksi, on komponentti, joka on tarkoitettu laajaan teollisuus- ja sovelluskäyttöön. COTSeista koostuvat sovellukset ovat aikaa myöten edullisempia kuin varta vasten kehitetyt järjestelmät. COTSeilla on laajat käyttökokemukset, mutta erityisesti turvallisuuteen liittyvien järjestelmäsovellusten haittana on käyttökokemustietojen vaikea saatavuus, mikä on edellytys vakuuttautumisessa COTS-pohjaisista järjestelmistä. Tutkimuksessa keskitytään luotettavuuden arviointiin sekä kolmannen osapuolen että luotettavuusominaisuuksien näkökulmasta.

Hyvän ohjelmiston kehitysprosessin ongelma on siinä, ettei se takaa hyvää ohjelmistoa. Erilaiset tuotteen arviointimenetelmät (mm. testit ja formaalit verifioinnit) eivät myöskään takaa hyvää ohjelmistoa. Ohjelmiston laadun evaluointi kohdistuu henkilö-, tuote- ja prosessiarviointiin. Niistä julkaisu keskittyy tuotteenarviointiin kolmesta hyvin erilaisesta näkökulmasta: sertifiointi pisteytysmenetelmällä, ohjelmiston testauksiin perustuvalla luotettavuustekniikalla ja riskienhallinnan tekniikalla.

Harju, Hannu. Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi. Osa 1 [Cost-effective design and assessment of dependable software. Part 1]. Espoo 2002. VTT Tiedotteita – Research Notes 2151. 114 p. + app. 15 p.

Keywords software dependability, safety integrity levels, reliability scoring, software reliability engineering, risk management process

Abstract

Software is increasingly being used in critical applications. Unlike most hardware failures, software failures are systematic and software faults may lie hidden for a long time before being revealed. This publication first introduces some affects that impact software reliability, and then three different aspects of software dependability: utilising integrity levels, dependability of COTS software, and warranty of software dependability.

Many standards for the development of safety-critical software and systems introduce the notion of safety integrity levels. It is a measure of the required likelihood that a safety function achieves its safety requirements. The approach of the safety integrity level has its obvious advantages, but also problems which make it inconvenience for utilising in practice. This publication contains a brief review of some widely referenced standards, to consider a context for utilising the approach for other dependability attributes than safety.

A commercially available (frequently called COTS) software component is one that has been developed for use in a variety of industries and applications. The system applications developed with the COTS software component are expected to be cheaper over the lifetime of the application than the custom designed plant application. The COTS system has a wider experience base. The disadvantages, especially for safety related systems, is that it may be difficult to get enough information to make a convincing acceptance argument for the COTS -software based application. The COTS software and systems constitued from them is considered from third parties viewpoint and from dependablity features as design, security, maintainability and concentrated failures.

A problem of good software development processes is that they do not guarantee good software. Different forms of product assessment (testing as well as techniques such as formal verification) are developed, but also they do not guarantee good software. Evaluation of software quality consists of three parts: personal, product and process evaluations. All of them is also needed for a dependability evaluation. The publication is concentrated in product certification from three different viewpoints: reliability scoring, software reliability engineering, risk management process.

Alkusanat

Tutkimushanke toteutettiin ABB Industryn, Fortumin, Teollisuuden Voima Oy:n ja VTT Tuotteet ja tuotannon yhteistyönä. Näiden yritysten lisäksi rahoituksesta vastasi Teknologian kehittämiskeskus (Tekes). Varsinaisen tutkimustyön hoiti pääasiallisesti VTT Tuotteet ja tuotanto. Tiedote on tutkimusprojektin "Kustannustehokas ohjelmistojen luotettavuuden suunnittelu ja arviointi, CERD" ensimmäinen osa.

Kiitämme kaikkia osallistuneita tahoja ja henkilöitä arvokkaasta panoksesta. Erityiskiitokset johtoryhmän edustajille Kari Rintalalle, TEKES, Jari Pesoselle, Teollisuuden Voima Oy, Jari Siitoselle ja Martti Välisuolle, Fortum, Jari Yli-Juutille, ABB Industry, Marja-Leena Järviselle ja Heimo Takalalle, Säteilyturvakeskus, Irmeli Lambergille, Innopoli Oy sekä Olli Ventälle ja Mika Koskelalle, VTT.

Hannu Harju

Sisällysluettelo

Symboliluettelo

BBN	Bayesian Belief Networks
CBR	Case-Based Reasoning
CBSD	Component Based Software Development,
CERD	Cost Effective Reliability Design
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
COTS	Commercial Off The Shelf
DIL	Dependable Integrity Level
DIT	Depth of Inheritence Tree
DLL	Dynamic Linking Library
EN	European Standard
EPRI	Electric Power Research Institute
FDA	Food and Drug Administration
IAEA	International Atomic Energy Agency
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standard Organisation
IV&V	Independent Verification and Validation
LOC	Level Of Concern
MCSE	Microsoft Certified Systems Engineer
MISRA	Motor Industry Software Reliability Association
MoD	Ministry of Defence
NASA	The National Aeronautics and Space Administration
NOC	Number Of Children
OTS	Off-The-Shelf
PEMS	Programmable Electrical Medical Systems

RAD	Rapid Application Development
RPN	Risk Priority Number
SEI	Software Engineering Institute
SPICE	Software Process Improvement and Capability dEtermination
STUT	Statistical Usage Testing
TET	Turvallisuuden eheystaso
TLJ	Turvallisuuteen liittyvä järjestelmä
UL	Underwriter’s Laboratory

1. Johdanto

Ohjelmistoala laajentaa reviiriään jatkuvasti. Tuskin on todennäköistä, että edes merkkejä supistumisesta olisi näköpiirissä. Ohjelmistoja hyödynnetään jo turvallisuuskriittisillä aloilla, joilla ohjelmiston virhetoiminta voi aiheuttaa joko vahinkoja ihmisille, ympäristölle, omaisuudelle tai kalliita käyttökeskeytyksiä. Turvallisuusalalla ohjelmiston kehitysprosessit ovat jo suhteellisen hyvin standardoituja. Osapuolet tunnistavat ohjelmiston rajallisuuden, eivätkä helposti sijoita sitä kohteisiin, joissa rajoja ylitettäisiin.

Mutta rajoja koetellaan. Ohjelmistojen toiminnat ja ominaisuudet monipuolistuvat sitä vauhtia, että myös turvallisuuskriittisiin kohteisiin on ehdolla ’rajat ylittäviä’ ohjelmistoja siitä yksinkertaisesta syystä, että muita ei ole saatavilla. Kaupalliset valmisohjelmistot ovat niistä hyvänä esimerkkinä. Ohjelmistoala on tunnetusti innovatiivista rajojen kokeilua, jossa pyritään hyödyntämään uutta kehitettyä teknologiaa, ja näköpiiriin ne avaavat mittaamattomat taloudelliset ja pätemisen mahdollisuudet.

Eräs asia on kuitenkin lapsenkengissään. Ohjelmiston luotettavuutta tai sen epävarmuutta ei edelleenkään kyetä riittävän hyvin mittaamaan riittävän pienin ponnistuksin. Päinvastoin: testaaminen vie yhä suuremman osan projektiajoista etenkin suuremmilla ja kriittisimmillä ohjelmistoilla. Ja usein testaaminen on ainoa käytännön keino ohjelmiston laadun ja luotettavuuden määrittämiseksi.

Selvää kehitystä on tapahtunut aivan viime vuosina sekä automaattisessa testauksessa että ohjelmistoprojektien riskienhallinnassa. Testaamisen kattavuuden lisääminen ja testausponnisteluiden vähentäminen on ollutkin jatkuvan kehittämisen kohteena. On myös ohjelmistoalan yrityksiä ja käyttäjiä, jotka säännöllisesti tekevät riskianalyysipohjaisia tarkasteluita ohjelmistoprojektin ja ohjelmiston vaarojen tunnistamiseksi ja niiden mahdollisilta ikäviltä vaikutuksilta välttymiseksi. Riskit luotettavuusmielessä eivät kuitenkaan ole olleet merkittävällä sijalla siten, että mitattaisiin luotettavuutta kehitysprojektin aikana joko kehittämisen ohjaamiseksi tai julkistamispäätöksen tukemiseksi.

Tämä julkaisu perustuu tutkimusprojektiin, jossa suunnataan ohjelmiston luotettavuustutkimusta ja haetaan tehokkaita ja taloudellisia tapoja arvioida luotettavuuden attribuutteja[1] sekä suunnitella ohjelmisto helposti arvioitavaksi. Tutkimustuloksissa viitataan myös ohjelmistoalan luotettavuuden koulutustarpeeseen, mikä ainakin vielä tällä hetkellä on aivan liian suuri.

[1] Toimintavarmuus, käytettävyys, turvallisuus, ylläpidettävyys ja tietoturva.

Tässä julkaisussa ohjelmiston luotettavuuden mittaamisongelmaa lähestytään kolmella teemalla:

1. Eheystasojen hyödyntäminen. Vastataan kysymykseen miten luokitella ohjelmistot sellaisiin luotettavuustasoihin, joihin kyetään määrittelemään juuri sille tasolle ominaiset luotettavuuden mittaamisen ja mittaamista edistävät menetelmät.

2. Valmisohjelmistojen ja niistä koostuvien sovellusten luotettavuus. Tarkastellaan kaupallisia ja yrityksen omia valmisohjelmistoja eri näkökulmista. Yhden näkökulman mukaan kyse on mustasta laatikosta, jonka luotettavuus on selvitettävä ilman informaatiota laatikon sisällöstä. Toisen näkökulman mukaan tulisi selvittää valmisohjelmiston riskit ja suojautua niiltä. Kolmannen mukaan valmisohjelmistot voivat myös olla hyvin laajoja ohjelmistokokonaisuuksia, joiden valmistamiseen tarvitaan perättäiskehittämistä sekä komponentteihin ja alustoihin perustuvia suunnittelumenetelmiä

3. Ohjelmiston luotettavuustakuut. Tutkitaan miten paljon (korjaamistiheyttä, sanktioita) yritys (valmistaja, toimittaja, käyttäjä) voi luvata tiedostaessaan ohjelmistonsa luotettavuuden täsmällisemmin. Teemaan liittyvät lainopilliset näkökulmat ja ehkä erityisesti niiden puuttuminen vielä niin nuorella toimialalla kuin ohjelmistovalmistus.

Ohjelmiston luotettavuus riippuu monesta tekijästä. Julkaisun aluksi, ennen yllä mainittuja erityisteemoja, tarkastellaan niistä muutamia merkittävimpiä. Ohjelmiston monimutkaisuus tulee aina olemaan keskeinen luotettavuuden suunnitteluun ja arviointiin vaikuttava tekijä. Organisaation ja ohjelmoijan kyvykkyys ovat vaikeasti mitattavia ominaisuuksia, mm. turvallisuudelle ei ole olemassa omaa kyvykkyyden arviointiin tähtäävää standardia. Kuitenkin ohjelmiston luotettavuuden mittaamisessa on aina jossakin määrin luotettava ohjelmiston suunnittelijoihin, testaajiin tai muihin kehitysprosessin osapuoliin.

2. CERD-tutkimusprojekti

CERD-kokonaisprojekti ”Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi” jakaantuu neljään osaprojektiin, jonka ensimmäisen osuuden tulokset raportoidaan tässä julkaisussa. Toisen osuuden tulokset raportoidaan vuoden 2002 aikana, kolmannen vuoden 2003 alkupuolella ja neljännen vuoden 2004 alussa.

Ohjelmiston laatua on lähestytty eri näkökulmista kehittämällä laatumalleja (mm. CMM, CMMI, SPICE, ISO 9000-3), laadun mittauksia, formaaleita menetelmiä, oliopohjaista suunnittelua, kielten ja ohjelmiston kehitysstandardeita jne. Kaikilla näillä lähestymistavoilla on etunsa ohjelmiston laatutason nostamisessa, mutta ne eivät ole kustannustehokkaita ratkaisuja. Vaikka testaaminen on laadun tärkein osoittamistapa, sillä on puutteensa. Kaikkialla pyritään testausta lisäämään, ja testaamisen suhteellinen osuus ohjelmistoprojektin kokonaisuudesta on koko ajan ollut kasvussa.

Luotettavuus on yksi laadun ominaisuuksista. Tutkimuksissa painotetaan luotettavuuden arvioinnin menetelmiä sekä ohjelmistotuotannon prosessien parantamista luotettavuuden arvioinnin helpottamiseksi. Näistä näkökulmista ohjelmiston luotettavuustutkimus on hyvin laajaa. Suomessa VTT on tutkinut ja kehittänyt ohjelmiston luotettavuus- ja riskianalyysipohjaisia tekniikoita. Tekniikat voidaan jaotella kahteen koetettuun tarkastelutapaan, ensinnäkin kvantitatiiviseen, jota edustavat mm. STUT (Statistical Usage Testing) ja BBN (Bayesian Belief Networks), sekä toiseksi riskianalyysipohjaiseen tarkasteluun, jota edustaa Tekesin teknologiaohjelman eräässä projektissa kehitetty arviointimenettely Tiira. Tekniikoita on sovellettu käytännön ohjelmistoprojekteissa. Vaatimus kustannustehokkuudesta tulee selkeästi esille Musan standardoimassa menetelmässä (Musa 1998) sekä kansainvälisessä IAEA:n tutkimusprojektissa (IAEA 2001).

Ohjelmiston luotettavuuden arviointi on vaikeata monestakin syystä, esimerkiksi

- yhdessä sovellusympäristössä luotettava ohjelma ei välttämättä ole luotettava toisessa
- yksinkertainenkin ohjelma saattaa koostua monimutkaisesta luotettavuusrakenteesta
- korjaaminen, versiointi ym. muuttaminen on altista virheille: miten arvioida yksinkertaisesti muutosten vaikutus luotettavuuteen arvioimatta samalla uudelleen koko ohjelmistoa
- luotettavuuden arviointi ilman käyttökokemuksia on usein tarpeellista, mutta siihen ei ole tiedossa käytännöllistä tapaa
- luotettavuutta tulisi kyetä arvioimaan myös ennen testauksia
- ei tiedetä riittävän tarkasti, mitkä tekijät vaikuttavat ohjelmiston luotettavuuteen.

CERD-hankekokonaisuuden tavoitteena on tutkia mahdollisuuksia kehittää kriittisillä aloilla käytetyistä täsmällisistä menetelmistä ja tekniikoista käytännönläheisiä työvälineitä ohjelmiston luotettavuuden suunnitteluun ja arvioimiseen myös vähemmän kriittisillä aloilla. Edelleen tavoitteena on järjestää ohjelmiston luotettavuusalan koulutusta yrityksille, korkeakouluille ja ammattikorkeakouluille. Tutkimuksen perusteemat ovat siten

- ohjelmiston luotettavuuden suunnittelu
- ohjelmiston luotettavuuden arviointi
- ohjelmiston luotettavuuden koulutuksen järjestäminen.

Tutkimuksen pääpaino on uusien ohjelmiston kehitysprosessien luotettavuusteknisessä hallinnassa. Näkökulma esimerkiksi ohjelmistoprosessien jatkuvaan parantamiseen ei ole laadun vaan luotettavuuden näkökulma. Kehitettävät työvälineet voivat olla mm. erilaisia deterministisiä sääntöjä koskien suunnittelua ja arviointia elinkaaren vaiheissa tai koskien suoraan tuotetta (esim. valmisohjelmistot). Työvälineet voivat myös perustua täsmällisen tai juuri oikean luotettavuuden periaatteeseen tai ne voivat perustua riskienhallinnan työvälineisiin. Työvälineet voivat myös tukea monia muita ohjelmiston laadunhallinnan elementtejä kuten laatujärjestelmiä, formaaleita määrittely-, suunnittelu- ja arviointimenetelmiä, testauksia, jne.

Työvälineillä tulisi kyetä arvioimaan ohjelmiston luotettavuus elinkaaren vaiheissa. Täsmällisellä tietämyksellä pystytään reagoimaan ohjelmistoprojektille vaadittaviin muutoksiin, esimerkiksi tavoitteiden muuttuessa markkinoiden vaatimuksista.

Vaikka luotettavuuden arvioinnin tulisikin tuottaa täsmällistä tietoa ohjelmiston virheettömästä toiminnasta tietyssä ympäristössä, arvioinnin tulee olla myös kustannustehokasta. Kustannustehokkuus riippuu saavutettavista säästöistä ja mm. seuraavista seikoista:

- vaadittava luotettavuustaso tunnetaan ja toteutetaan ohjelmisto juuri tähän oikeaan tasoon
- käytetään sopivaa joukkoa hyväksi todettuja luotettavuuden suunnittelu- ja arviointimenetelmiä
- työvälineiden käyttöaika on kohtuullinen
- työvälineen koulutusajan on oltava kohtuullinen
- laaditut menetelmät ovat yleismenetelmiä, jotka kohdistuvat sekä suunnitteluun, arviointiin että koulutukseen, ja niiden tulisi soveltua kaiken tyyppisille ohjelmistoille.

Edellä mainittujen perusteemojen lisäksi CERD-projektiin kuuluu erityisteemoja, jotka ovat luonteeltaan esitutkimustyylisiä. Erityisteemoissa kartoitetaan tutkimuksen tarve sekä suuntaviivat.

Ensimmäinen tutkimusosuus koostuu seuraavista, tässä julkaistuista erityisteemoista:

1. Eheystasojen hyödyntäminen
2. Valmisohjelmistojen ja niistä koostuvien sovellusten luotettavuus
3. Ohjelmiston luotettavuustakuut

Myöhemmät tutkimusosuudet koostuvat ainakin seuraavista erityisteemoista:

4. Valmiudet uusien luotettavuusmenetelmien käyttöönottoon
5. Testausautomaatio ohjelmiston luotettavuuden ilmaisijana
6. Ohjelmiston virhemekanismit
7. Ohjelmiston luotettavuuden mittaaminen
8. Kehitysprosessien hajauttaminen luotettavasti
9. Ohjelmiston ylläpidettävyys
10. Formaalien menetelmien integrointi luotettavuusmenetelmiin.

3. Ohjelmiston luotettavuuden arviointi: luotettavuustekijät

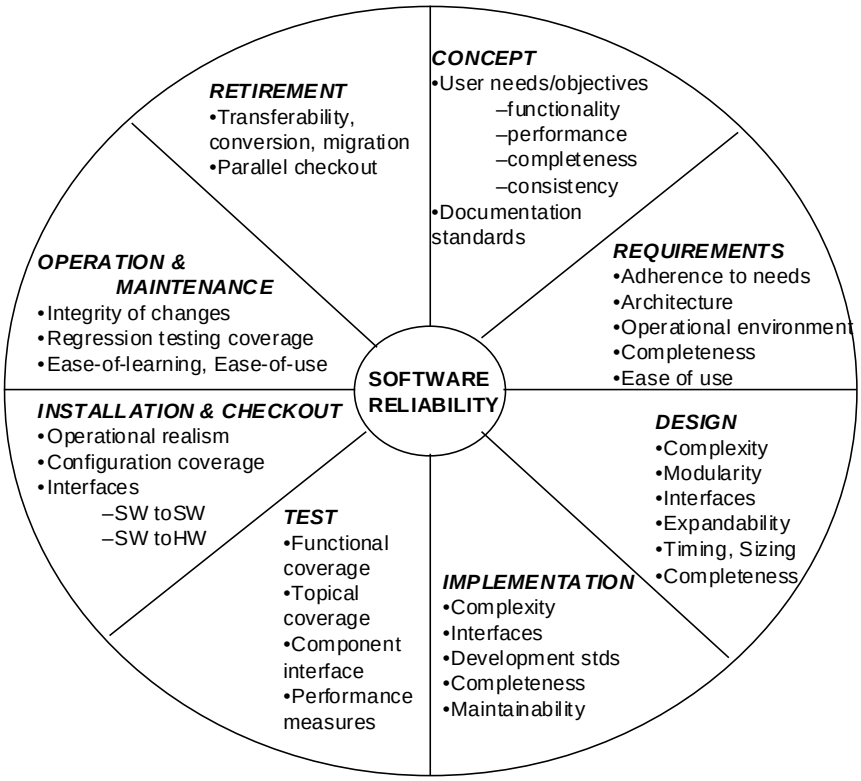
Kirjallisuudesta löytyy tutkimuksia, jotka eri näkökohdista priorisoivat ohjelmiston luotettavuuteen vaikuttavia tekijöitä. Haastattelututkimuksessa Zhang ja Pham (2000) asettavat 32 tekijää tärkeysjärjestykseen. Luotettavuustekijät on esitetty prioriteettijärjestyksessä liitteen A taulukossa A1.

Zhangin ja Phamin tutkimukseen osallistui 24 ohjelmistokehittäjää 13 yhdysvaltalaisesta ohjelmistoyrityksestä. Henkilöt edustivat kauttaaltaan kaikkia ohjelmistokehitysrooleja managereista, systeemi-insinööreistä ohjelmoijiin, testaajiin ja muihin ohjelmistoalan tutkijoihin ja kehittäjiin. Ohjelmoijat olivat suurimmassa ryhmässä (40 %). Vaikka mukana oli yrityksiä, jotka toimivat turvallisuuskriittisillä aloilla ja lisäksi yrityksiä, jotka valmistivat ohjelmia omaan käyttöön, kaikki valmistivat kaupallisia ohjelmistoja.

Vaikka em. tutkimus on suhteellisen pieni, se antaa kuitenkin aika hyvän lähtökohdan itse kullekin arvioida ohjelmiston luotettavuuteen vaikuttavia parannuskohteita omassa yrityksessään. Tutkimus on tieteellisin perustein tehty, mikä edesauttaa lukijaa perustelemaan itse oman tärkeysjärjestyksensä ohjelmiston luotettavuuteen vaikuttavista tekijöistä. Empiirisesti tutkittiin vaikuttivatko ohjelmistokehittäjien eri roolit, ohjelmistoala (turvallisuuskriittinen, kaupallinen, sisäinen ohjelmistovalmistus), ja tekijöiden seurausvaikutusten erilaisuus haastateltavien mielipiteisiin. Rooleilla oli tilastollista merkitystä mielipiteisiin ohjelmistoluotettavuuteen vaikuttavista tekijöistä. Haastateltavan ohjelmistoalalla ei ollut, mihin ilmeisesti vaikutti se, että kaikki tutkimukseen osallistuvat yritykset valmistivat myös kaupallisia ohjelmistoja. Luotettavuustekijöiden seurausvaikutusten erilaisuudella, mm. mahdollisten vahinkojen suuruudella, ei katsottu olevan merkitystä. Edelleen otettiin huomioon korrelaatiotarkastelut kaikkien 32 tekijän riippumattomuudesta. Tutkimusmenetelmään ja -hypoteeseihin ei tässä yhteydessä ole syytä tämän tarkemmin paneutua. Niistä kiinnostunut lukija voi tekstissä annettujen yksityiskohtaisten ohjeitten pohjalta tehdä vastaavan tutkimuksen vaikka omassa yrityksessään. Käsitellään tässä lyhyesti tutkijoiden aikaansaamaa järjestystä ohjelmistoluotettavuuteen vaikuttavista tekijöistä.

Tutkimuksen mukaan kolmestakymmenestäkahdesta merkittävästä ohjelmiston luotettavuuteen vaikuttavasta tekijästä (Liite A: Taulukko A1) kuusi tärkeintä ovat ohjelmiston monimutkaisuus, ohjelmoijan taito, testausten kattavuus, testausponnistelut, testausympäristö ja ohjelman spesifikaation muutostiheys. Merkille pantavaa on, että 32 tekijän joukkoon on merkitty testauksia, ei staattisia analyyseja. Ilmeisesti testaukset ovat yleismerkityksessään sisältäen myös yleensä kaikki analyysimenetelmät.

Ohjelmointikieli on järjestykseltään vasta 24. sijalla tutkimuksen mukaan. Empiirinen vertailu (Prechelt 2000) osin tukee tätä käsitystä havaitsemalla C/C++ -kieliset ohjelmat hivenen epäluotettavammiksi kuin Javalla ja nk. skriptisillä kielillä (Perl, Python, Rexx, Tcl) tehdyt ohjelmat. Virheellisyys rajoittui kuitenkin vain muutamaan ohjelmaan, eikä yleistämistä epäluotettavuudesta voitane siten tehdä.



Kuva 1. Ohjelmiston luotettavuuden rakentuminen (IEEE 982.2).

Ohjelmiston korkea luotettavuus rakentuu usean laatuattribuutin soveltamisesta jokaisessa elinkaaren vaiheessa. Standardi IEEE 982.2 esittää yhden esimerkin tästä lähestymistavasta (kuva 1).

Käsitellään seuraavaksi joitakin merkittävimpiä luotettavuuteen vaikuttavia tekijöitä ohjelmiston luotettavuuden rakentumisessa. Tarkastellaan tekijöiden korreloituvuutta CERDin 1. vaiheen erityisteemojen kanssa ja selvitetään lisäksi, olisiko tekijöistä uusiksi erityisteemoiksi tai tutkimuskohteiksi.

3.1 Ohjelmiston monimutkaisuus

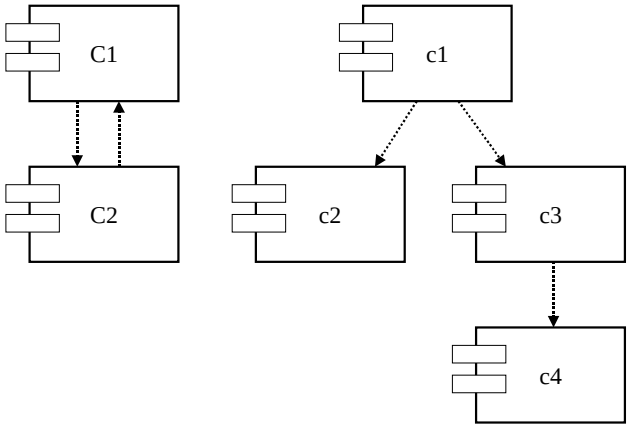
Ohjelmiston monimutkaisuus on yleisesti todettu merkittävimmäksi luotettavuuteen vaikuttavista tekijöistä. Yhtä mieltä ei kuitenkaan olla siitä miten sitä voidaan mitata. Mittausvälineitä on ehdotettu melkoinen määrä muutaman viime vuosikymmen aikana (luettelo: Zuse 1990), jopa niin paljon, että niistä on kirjoitettu yhtä paljon kirjoja (40 teosta) metriikka-alan gurun Fentonin mukaan (Fenton & Neil 1999) kuin koko ohjelmiston muusta kehittämisestä. Välineistä tunnetuimmat ovat mm. McCabe'in (1996) ja Halsteadin (1977) metriikat, mutta eräät viimeaikaiset tutkimukset osoittavat, että pelkkä ohjelmiston koko kuvaa lähes yhtä hyvin (tai huonosti) ohjelmiston monimutkaisuutta[2] kuin nämä. Edellä mainitussa tutkimuksessa haastateltaville annettiin ohjeeksi määritellä monimutkainen ohjelma 20 kilorivin in suuruiseksi. Tämäkin yksinkertainen kriteeri riitti monimutkaisuuden asettamiseksi merkittävimmäksi luotettavuustekijäksi.

Käytännössä yksinkertaiset monimutkaisuusmitat ovat suosiossa. Ne ovat helppoja laskettavia, muiden ollessa lähinnä sisäpiirin välineitä tarvittavine osaamisineen ja työlyyksineen. Tämä tiedostaen ollaan kehittämässä (mm. Fenton & Neil 1999) projektipäällikölle päätöstä tukevia välineitä. Ne perustuisivat Bayesin verkkoihin, joiden monimutkaisuus kätketään käyttäjiltä. Niissä otetaan huomioon monia kehitys- ja testausaikaisia tekijöitä. Koska ne perustuvat Bayesin verkkoihin, on oletettavaa, että ne käsittelevät päätöksenteon epävarmuustekijöitä ja evidenssien yhdistämisiä.

Edellä käsitellyssä Zhangin ja Phamin tutkimuksessa ohjelmiston monimutkaisuus perustui haastateltavien mielleyhtymiin monimutkaisuudesta, siis se saattoi perustua pelkästään ohjelmiston kokoon, kuten tutkijat heitä opastivat. Vaikka jotkut kirjoitukset tukevatkin käsitystä siitä, että ohjelmiston koko antaa lähes yhtä hyvän tuloksen kuin monimutkaisuusmetriikat, huomattava osa ohjelmiston suunnittelusäännöistä kohdistuu nimenomaan monimutkaisuuden vähentämiseen. Sellaisia ovat mm. rakenteellisuus, modulaarisuus ja riippuvuuksien vähentäminen. Esimerkiksi Jaaksi et al. (1999) pureutuvat näihin sääntöihin suosittaessaan syklisen riippuvuuden eliminoimista kuvan 2 esittämällä tavalla. Jo kuvankin esityksen mukaan ohjelmiston koko kasvaa, ellei sitten syklisyyden ohjelmointi vie suhteettomasti alaa. Syklisen ohjelmiston tekeminen voi olla vaikeampaa kuin kuvan mukaisen hierarkkisen ohjelmiston, ja sitä tietä monimutkaisuus yhdessä ohjelmoijan taidon ja välineiden kautta vaikuttaisi enemmän ohjelmiston luotettavuuteen kuin laaja hierarkkinen vastaavuus. Ohjelmiston koko voi vastata monimutkaisuutta, mutta ei luotettavuutta.

---

[2] Fenton ja Neil (2000) ovat tästä eri mieltä. He väittävät, että kehittyneitä mittareita ei käytetä, koska ne ovat työläitä. Ne ovat vain suljetun, etupäässä tutkimusyhteisön välineitä sillä datan saatavuus on hankalaa.



Kuva 2. Syklisen ohjelman (C1, C2) hajauttaminen (c1..c4) (Jaaksi et al. 1999).

Vaikka ohjelmiston monimutkaisuudesta on kirjoitettu paljon, monimutkaisuuden ja luotettavuuden välinen suhde on jäänyt vähemmälle. Monissa teoksissa luotettavuus arvioidaan estimoimalla ohjelmistovirheiden lukumäärää eikä mukaan ole otettu kaikkia ohjelmiston käyttötilanteita (Ottenstein & Fodsick 1979, Basili & Perricone 1984, Takahashi & Kamayachi 1989). Vaikka tilastollisesti onkin todettu ohjelmiston monimutkaisuuden korreloivan virhemäärän kanssa, korreloituvuus on yksittäisestä ohjelmistosta riippuva.

Myös eksplisiittisesti luotettavuuden kasvumallit huomioon ottavat mittaustavat eivät ole vakuuttavia (mm. Munson & Khoshgoftaar 1991). Niistä puuttuvat sekä kokemusperäiset että validoivat tiedot (Laprie 1998).

Tuotepohjaisen mittaamisen integroiminen prosessimittoihin on myös ollut tutkimuksen kohteena (mm. Takahashi & Kamayachi 1989). Niissä virheiden kokonaismäärä oli validoinneissa havaittujen virheiden määrä, mikä ei vastaa jäännösvirhemäärää. Myös suhde validoinneissa löydettyjen virheiden lukumäärän ja jäännösvirheiden lukumäärän välillä jää vaikeasti havaittavaksi.

Koska ohjelmiston monimutkaisuus on eittämättä yksi merkittävimmistä ja ehkä merkittävin ohjelmiston luotettavuuteen vaikuttava yksittäinen tekijä, tulisi nimenomaan luotettavuuden tutkimus- ja kehitysalalla tarkastella käytännönläheisiä, yksinkertaisia ja projektitoimintaan upotettuja ohjelmiston monimutkaisuuteen perustuvia työvälineitä. Uusia mittareita tai mittaamistapoja tuskin tarvitsee enää kehittää, vaan tulisi löytää olemassa olevista sopivat.

Ohjelmiston kehitysprosessit ovat tulossa yhä työläämmiksi ja kalliimmiksi ohjelmiston monimutkaisuuden kasvamisen myötä. Monimutkaisuuden alentamiseen tähtäävät työ-

kalut (mm. mittausvälineet) eivät saisi lisätä ennestäänkin kasvavaa ja komplisoituvaa ohjelmistonkehitystä.

Em. tutkimuksessa (Zhang & Pham 2000) otettiin huomioon myös luotettavuustekijöiden korrelaatiot (liite A: taulukko A2) eli miten ne riippuivat muista luotettavuustekijöistä. Ohjelmiston monimutkaisuus korreloi kehitystiimin koon kanssa, mutta on selvää, että se korreloi myös ohjelmoijan taidon kanssa jopa niin, että on vaikea eritellä monimutkaisuuden luotettavuusvaikutuksia ja ohjelmoijan tai ohjelmistokehittäjän taitoa.

Viittauksia ohjelmiston kehitysprosessien parantamiseen on kuitenkin olemassa. Pressman (1997) puoltaa yksinkertaisia mittauksia[3], koska käytännön työskentelyssä niiden on todettu antavan johdonmukaisen ja objektiivisen menettelytavan laadun arvioimiseen ohjelmistoprosessin ensi vaiheissa.

Voidaan kuvitella ohjelmiston sisäisten rakennetta kuvaavien metriikoiden vastaavan ohjelmistotuotteen ja ohjelmiston kehitysprosessin ominaisuuksia, mutta huolimatta laajasta metriikkakirjallisuudesta teoreettista näyttöä tälle yhteydelle ei ole saatavilla. Metriikkojen validoitavuuteen ei ole kiinnitetty riittävästi huomiota ja siten niillä ei ole merkittävää roolia riskien vähennyksessä.

Fentonin (1995) mukaan ohjelmistoattribuuteilta puuttuu yleensä hyvä empiirinen malli. Mittojen toimivuutta yritetään arvioida vertaamalla niiden ennustamia tuloksia todellisiin tai tekemällä korrelaatioanalyysi. Prosessissa, tuotteessa ja ympäristössä on monia tekijöitä, joita ei voi kokeessa vakioida, joten ainoaksi keinoksi jää nähdä mittaluvut satunnaislukuina, otoksena, ja tehdä aineistolle tilastollinen analyysi.

Mitoista ei ole hetkellistä hyötyä, sillä yksi mittaustapahtuma ei vielä vaikuta mihinkään. Metriikoita voi käyttää laadunosoituksessa hyväksi vasta, kun tuotteesta tai prosessista on kerätty aineistoa, jolle voidaan tehdä analyysi: mittaukset on tehty vakioidussa ympäristössä, mittaajan tai ympäristömuutosten vaikutukset on kompensoitu, luonnollisen vaihtelun amplitudi tunnetaan.

Monimutkaisuudesta aiheutuvat luotettavuusongelmat ovat piilossa olevaa tietoa. Jollakin tai jossakin on sellaista merkittävää tietoa, jonka puuttuminen voi aiheuttaa ja on aiheuttanutkin kalliita vahinkoja (mm. Ariane 5, Mars-luotain) tai jopa ihmishenkien menetyksiä (Therac 25).

[3] Yhdistetään mittauksiin teknisesti niin monia ominaisuuksia, että väkisinkin syntyy ristiriitoja (Fenton 1994).

3.2 Ohjelmoijan taito

Ohjelmoijan taito on em. tutkimuksen lisäksi monen muunkin lähteen (Zeigler 1995, Prechelt 2000) mukaan merkittävimpiä ohjelmiston luotettavuuteen vaikuttavia tekijöitä. Tärkein ohjelmistotuotannon tekijä on ohjelmoijan tuottavuus, mikä korreloi sekä taitavuuden, työn hankaluuden, asiakassuhteen luonteen että työkalujen ja teknologian saatavuuden kanssa. Rahan tuhlaaminen kaikkiin näihin tekijöihin vähentää ohjelmiston kehityskustannuksia (Mills 1988).

Taitavat ohjelmoijat saattavat olla Pressmanin (1997) tietojen mukaan jopa kaksikymmentä kertaa tehokkaampia kuin ohjelmoijat keskimäärin. Siten jo tuottavuuden kannalta on tärkeää saada pidettyä heidät talossa. Heidän lähdettyään menetetään sekä tuottavuudessa että ohjelmiston luotettavuudessa. Yrityskulttuuri eli tapa määritellä, kuinka asiat tehdään, voi viehättää mestariohjelmoijia. Aina viehätyksen kohteina eivät ole tiukat laatuohjeet, jotka auttavat yritystä parempien ohjelmien valmistuksessa kokonaisuudessaan.

Ohjelmiston kehittäjällä pitää myös olla silmää ymmärtää vaatimukset ja yksinkertaistaa ongelmaa. Hyvän ohjelmoijan taitolajeihin kuuluu monimutkaisuuden vähentäminen jo varhaisessa vaiheessa ohjelmistokehitystä (mm. protoilu). Boehm (1981) havaitsi, että ohjelmoijan tai tiimin kyvykkyys yhdessä tuotteen monimutkaisuuden kanssa vaikutti merkittävimmin sekä kustannuksiin että tuottavuuteen.

CERDin yhtenä tavoitteena on lisätä ohjelmiston luotettavuusalan *koulutusta* yrityksissä ja opinahjoissa. Ohjelmistokehittäjien koulutuksesta ollaan sitä mieltä (Zeigler 1995), että ohjelmistokehittäjät kouluttautuvat ensi sijassa työssään. Oppiminen tapahtuisi parhaiten noudattamalla jo olemassa olevia suunnittelu- ja koodaussääntöjä. Zeiglerin mukaan tietoa tarvittaisiin ennen kaikkea virheiden käsittelyyn. Ohjelmiston luotettavuuteen voi myös kouluttautua työn parissa edellyttäen, että on olemassa hyviä käsikirjoja, jotka opastavat *luotettavan ohjelmiston suunnittelua ja arvioimista* tarvetilanteissa sekä eri kehitysprosessin vaiheissa.

Tärkeydestä huolimatta ohjeita ohjelmistokehittäjän kyvykkyyden mittaamiseen ei ole mitenkään liikaa. Standardi EN-IEC 61508 esittää formaalit pätevyysvaatimukset ohjelmistokehittäjille, jotka toimivat turvallisuuteen liittyvien järjestelmien parissa. On oltava alan mukaiset tiedot, koulutus ja kokemus. Kyvykkäät ohjelmistokehittäjät työskentelevät hyvin, nopeasti ja virheettä.

Ohjelmistokehittäjien sertifioimisesta puhutaan aina silloin ja tällöin. Henkilösertifiointi on kuitenkin suhteellisen vaikeaa, koska sen perusteoria puuttuu. Asetetut vaatimuksetkin, joihin edellä jo viitattiin, koskevat lähinnä keskitasoista kyvykkyyttä. Tämän ky-

vykkyyden ylittävillä taidoilla on kuitenkin huomattava osuus ohjelmiston luotettavuutta parannettaessa. Erityisteemassa *Eheystasojen hyödyntäminen* (ks. luku 4) tarkastellaan kyvykkyysvaatimusten asettamista eri luotettavuuden ja turvallisuuden eheystasoille.

Kyvykkyys liittyy ostajan kannalta erityisteemaan *Valmisohjelmistojen luotettavuus* (ks. luku 5) lähinnä organisaatiotasolla. Kyvykkyyden mittaaminen on kuitenkin samantasoinen ongelma oli sitten kyse yrityksestä tai yksittäisestä henkilöstä tai projektitiimistä. Laatusertifikaateilla osoitetaan nykyisin koko yrityksen tasoa tehdä hyviä ohjelmistoja. Sertifikaattien laatutaso kuitenkin vaihtelee, eikä sellaisen myöntäminen välttämättä ilmaise ohjelmiston korkeaa turvallisuuden eheyttä. Tarvitaan ensi sijassa ohjelmistotuotteen arviointia vaikka sertifikaatteinakin. Aihetta sivutaan yhdessä henkilö- ja prosessisertifioinnin kanssa erityisteemassa *Ohjelmiston luotettavuustakuut* (ks. kappale 6.2.1).

3.3 Testausten kattavuus

Tutkimus (Zhang & Pham 2000) asettaa testauksen kattavuuden kolmannelle tilalle priorisoitaessa ohjelmiston luotettavuudelle merkittäviä tekijöitä. Testauksilla tarkoitetaan myös staattisia analyyseja normaalien dynaamisten testausten lisäksi. Kattavuudella täydellisyyden lisäksi tarkoitetaan myös merkittävien asioiden testaamista. Ei siis yksin pyritä testaamaan mahdollisimman paljon vaan myös järkevästi. Dynaamisten testausten kattavuus on mahdollisimman laajan syötekombinaation testaamista. Lisäksi syötesekvensseillä on merkitystä erityisesti hajautetuissa reaaliaikaisissa järjestelmissä, joissa pienetkin syötesekvenssien muutokset tulee alistaa testauksille. Etenkin näissä testimäärät kasvavat suunnattomasti.

Luotettavuuden ja turvallisuuden analyysit sekä yleensä riskienhallinta ja riskianalyysit tukevat testien kohdentamista. Niissä tarkastellaan, mitkä ovat luotettavuuden tai turvallisuuden kannalta merkittävät tarkastelukohteet, ja sen jälkeen kohdistetaan suunnittelu-, toteutus-, arviointi- ja testausponnistuksia niihin kohteisiin.

Kvalitatiivisten luotettavuusanalyysien kattavuus hoidetaan ottamalla tarkasti huomioon tarkastelunäkökohdat. Niinpä esimerkiksi ohjelmiston vika-, vaikutus- ja kriittisyysanalyysin vioittumistapojen valinta ja riittävän täydellinen läpikäynti kaikissa ohjelmiston kehitysvaiheissa kuvaavat analyysin kattavuutta yhdessä syiden, seurausten ja riskien hallinnan keinojen kanssa. Osoitettaessa ohjelmistopohjaisen järjestelmän luotettavuutta analyysien kattavuuden osoittaminen on keskeisessä asemassa. Kolmannen osapuolen on analyysiraporteista saatava nopeasti hyvä käsitys analyysien täydellisyy-

destä olematta itse edes ohjattavan kohteen tai siihen liittyvän ohjelmistopohjaisen järjestelmän asiantuntija.

Viite (Harju 2000) esittelee erään ohjelmistojen luotettavuusanalyysin, joka soveltuu kaikkiin ohjelmistoihin, joiden virheetön toiminta on valmistajalle ja käyttäjälle tärkeää. Analyysikattavuus yhdessä testausten kattavuuden kanssa tulee menetelmässä selvästi esille. Menetelmää on myös käytännön ohjelmistoprojekteissa koekäytetty. Jatkotutkimuksen kannalta merkittävintä on keskittyä kattavuuden järkiperäistämiseen siten, että voitaisiin paremmin jo ensimmäisestä kerrasta keskittyä tietylle ympäristölle ominaisiin vikatapauksiin ja niiden tulkintoihin.

Testausten kattavuus esitetyssä laajemmassa tulkinnassaan liittyy läheisesti CERDin erityisteemaan ”Eheystasojen hyödyntäminen”. Eheystasojen sisältämät suunnittelu- ja muut säännöt ja suositukset mm. verifioinnille ja validoinnille tulee tehdä eheystasoa vastaavalla kattavuudella. Asia on monitahoinen. Yksittäisen esim. diagnostiikan kattavuusmenetelmän kohdalla ei ole päästy selkeään tulkintaan siitä, mitä kattavuus voisi tarkoittaa yhdessä muiden virheen ilmaisukeinojen kanssa. Riittävän ja juuri oikean luotettavuuden osoittamismielessä kyse on kokonaismenetelmien kattavuudesta, ei siis pelkästään analyysien ja testien. Suunnittelu- ja laatusäännöt olisi myös osattava ottaa huomioon eheystason mukaisesti.

Testausten kattavuus liittyy valmisohjelmistojen luotettavuusteemaan erityisesti silloin, kun COTSien luotettavuus pitää selvittää dynaamisilla testauksilla. COTSien yhdistäminen järjestelmäksi mahdollisesti räätälöityjen ohjelmistojen kanssa tekee järjestelmästä helposti laajamittaisen, mikä vaatii tuntuvaa testaamista luotettavuuden selvittämiseksi riittävällä varmuudella. Laajojen järjestelmien testaamisessa on lisäksi omat ongelmansa:

- Reaaliaikaominaisuuksien testaaminen riippuu toteutuksesta (kovo oltava mukana). Järjestelmä on testattava kokonaisena, mikä entisestään kasvattaa testaustarvetta. Automaattiset testauslaitteet ovat välttämättömiä testitarpeesta selviämiseksi.
- Testauslaitteet huolehtivat testisyötteistä, ajastuksen hallinnasta ja tulosteiden valvonnasta. Testilaitteilta vaaditaan hyvin korkeaa luotettavuutta.
- Testauslaitteiden suoritus perustuu satunnaislukugeneraattoreihin.
- Testisekvenssien vähentämiseksi testaamista on systemaattisesti kohdennettava tärkeille alueilla.

3.4 Testausponnistelut

Zhang & Phamin (2000) tutkimus asetti testiponnistelut merkittäväksi luotettavuustekijäksi. Edellä mainittu Precheltin (2000) julkaisema tutkimus vertailee seitsemää ohjelmointikieltä mm. niiden vaatiman kokonaistyöajan osalta. Päinvastoin kuin joissakin muissa lähteissä (Jones 1996, Boehm 1981) Precheltin tutkimuksen mukaan kokonaistyöaika vaihtelee eri kielillä. C, C++ ja Java ovat selvästi työläämpiä kuin skriptiset kielet Perl, Python, Rexx ja Tcl, vaikka niissä ohjelmakoot ovat puolet skriptisten ohjelmien koosta. Eroja eri kielten virhealttiudessa Prechelt ei havainnut. Tämän kaltainen empiirinen tutkimustyö on kuitenkin vaikeasti validoitavissa, sillä siihen vaikuttavat monet tekijät, mm. ohjelmoijan kyvykkyys, dokumentoinnin vaatima aika ja täsmällisyys sekä työtehtävien ja työolosuhteiden erilaisuus.

Testaamista joudutaan jatkuvasti lisäämään ohjelmistoprojekteissa. Automaattisista testaustyökaluista on varmasti hyötyä laajemmissa ohjelmistoprojekteissa, mutta hankaluutensa ja työläytensä on niissäkin. Testimäärittelyt ja valmistelut on hoidettava edelleen. Dustin et al. (2000), jotka kirjassaan esittävät monipuolisen automaattisen testaamisen elinkaarimallin, ovat sitä mieltä, että automaattisen testaamisen kehittäminen on aikaa vievää, eikä ensimmäisellä käyttökerralla vielä tuo säästöjä verrattuna normaaliin manuaaliseen testaamiseen. Myöhemmin, testauskokemuksen kerääntyessä, automaattiset testaamiset säästävät aikatauluissa, käsikirjojen käytössä, jokapäiväisissä ja toistettavissa töissä, jotka ovat työläitä ja virhealttiita.

Verrattuna manuaaliseen testaamiseen automaattinen testaaminen helposti lisää ohjelmistoprojektin kompleksisuutta, johon testausryhmällä ei ehkä ole aikaisempaa tuntumaa. Testaussuunnittelu vaatii komentokielisten ohjelmien kirjoittamista, mikä saattaa olla ryhmälle uutta, tai ehkä vain muutamat ryhmän jäsenet osaavat ohjelmointia. Vaikka testityökalut olisivatkin tuttuja, ne eivät välttämättä sovellu uusiin ohjelmistoprojekteihin.

Edellä mainittu testaamisen painottaminen riskianalyysipohjaisilla menettelyillä helpottaa asiaa, mutta sillä on samat ongelmat kuin automaattisilla testauksillakin. Kompleksisuus lisääntyy ja näkyy uusina opeteltavina työtapoina. Varhain tehdyt painotukset kuitenkin vähentävät virhealttiutta ja testaustarvetta sekä vaatimusten toteuttamista määrittelyssä, suunnittelussa ja koodauksessa. Virheiden vähentyminen merkitsee kustannusten, ajankäytön ja uudelleen tehtävän työn vähentymistä.

Testien ja analyysien kohdistaminen ja automatisointi, ”kerralla valmista”, ja juuri oikeaan luotettavuuteen tähtääminen ovat keskeiset elementit pohdittaessa, miten luotettavuustekniikka voisi tukea testausponnisteluiden vähentämistä. Nämäkin toimenpiteet vaativat huolellista suunnittelua, hyvin määritellyn ja strukturoidun prosessin sekä pätevät ohjelmistokehittäjät.

3.5 Ohjelman spesifikaation muutostiheys

Ohjelmiston vaatimusten tiheä muuttaminen oli haastattelututkimuksen mukaan kuudenneksi merkittävin ohjelmiston luotettavuutta vaarantava tekijä. Erityisesti perättäiskehittäminen on virhealtista. Siinä uuden tuotteen konstruointi tapahtuu lisäämällä ominaisuuksia ja toimintoja vanhaan tuotteeseen. Lisäominaisuudet ensisijaisesti muuttavat arkkitehtuuria ja onnistuakseen myös vaatimusmäärittelyä on muutettava. Virhealttius ilmenee jo testaus- ja ylläpitovaateiden kasvamisena.

Ohjelmistoon kohdistuvat vaatimukset muuttuvat jatkuvasti. Yleisesti kuvitellaan, että koska ohjelmisto on joustava, vaatimusten muuttaminenkin sekä edelleen toteuttaminen ja testaaminen ovat luotettavuuden kannalta joustavia toimenpiteitä. Muuttamisen vaikutukset riippuvat siitä missä vaiheessa ohjelmiston elinkaarta muutokset on tehty. Vaatimusten muuttaminen määrittelyvaiheessa on vielä suhteellisen helppoa, toteutus- tai testausvaiheessa (puhumattakaan julkistamisen jälkeisistä vaiheista) muutokset toimintoihin, suorituskykyyn, käyttöliittymiin ja muihin ominaisuuksiin ovat kalliita ja vaikeasti toteutettavissa, mikä varmasti on myös virhealtista heikentäen ohjelmiston luotettavuutta.

Laajoille ja kriittisille ohjelmistoille muutosten laadukas hallinta on välttämätöntä. Korkeaan luotettavuuteen johtavaan muutosprosessiin kuuluu useita toimenpiteitä. Muutosprosessiin kuuluvat mm. muutosmääräysten, katselmus- ja testauskriteerien laatimiset, muutosten toteuttaminen, katselmointi, testien suunnittelu, laadunvarmistus ja testaaminen. Näiden lisäksi muutostoimenpiteisiin kuuluu monia byrokraattisilta kuulostavia, mutta tehokkaita virhealttiutta vähentäviä toimenpiteitä.

Tämän kohdan alussa viitattiin perättäiskehittämisen malliin, joka on kehitetty erityisesti vastaamaan informaatioteknologian nopeasti muuttuvia markkinatarpeita. Tarkasteltaessa vaatimusten muutostiheyden merkittävyyttä ei välttämättä tarvitse käsitellä nykyaikaisia yhä monimutkaistuvia kehitysmalleja, vaan yksinkertaisia, historiallisia malleja esimerkkeinä perinteinen vesiputousmalli ja spiraalimalli (Boehm 1988).

Vesiputousmallissa tavoitteena on tehdä kerralla valmista. Uudelleen tehtävä työ minimoidaan vain vähäisillä takaisinkytkennöillä. Mutta käytäntö osoitti nopeasti, että läheskään kaikissa projekteissa ei vaatimuksia kyetä määrittelemään riittävän täsmällisesti projektin alussa. Hallitsemattomat riskit toivatkin mallille huonon maineen. Päätöksiä joudutaan tekemään liian aikaisin liian vähin tiedoin. Riskien vähentämiseksi kehitettiin spiraalimalli, jossa useaan spiraaliketjuun analyyseja, prototyypittämistä ja spesifiointia liitettiin jatkoksi riskianalyysit. Ne käsittelevät ja vähentävät riskejä siten, että viimeisen spiraaliketjun päätyttyä ei enää olisi merkittäviä riskejä. Projektin lopussa valmis oh-

jelmisto olisi siten riskitön. Kuitenkin vaatimusten jatkuva muuttaminen kiristää aikataulua ja luotettavuus laskee.

Kehitysprosessissa RAD[4] (Boehm 1999) vaatimuksia pidetään liikkuvina kohteina koko kehitysprojektin ajan. Erillinen katselmusryhmä, joka koostuu asiakkaista, käyttäjistä ja kehittäjistä, priorisoi toiminnalliset vaatimukset ja kohdistaa resurssit.

Kuvaan 1 viitaten luotettavuuteen vaikuttavia tekijöitä ovat vaatimusten rakenteellisuus, täydellisyys ja toteuttamisen helppous. Vaatimusten määrittelijän ja toteuttajan (suunnittelu ja koodaus) välillä pitää olla hyvä yhteisymmärrys, mikä yleensä johtaa ainakin korkeissa luotettavuusvaatimuksissa täsmällisen vaatimusmäärittelyn laatimiseen. Yhteisymmärrys alkuperäisten tavoitteiden kanssa, jotka tulevat asiakkaalta ja käyttäjältä, on myös ensiarvoisen tärkeää. Keskustelut tässä yhteydessä vähentävät keskustelun tarvetta myöhemmissä vaiheissa.

Standardeista on hyötyä täsmällisten ja kaikille osapuolille ymmärrettävien vaatimusmäärittelyjen laatimisessa. Sellaisia ovat tehneet useat standardoimisjärjestöt (mm. IEC, IEEE, DOD ja NASA).

[4] Rapid Application Development

4. Eheystasojen hyödyntäminen

Monet turvallisuuskriittiset standardit tukeutuvat käsitteeseen turvallisuuden eheystaso. Se on suunnittelua helpottava menettelytapa ja luokittelu, josta hyötyvät sekä järjestelmän hankkijat että valmistajat. Valmistajat tuottavat kaupan hyllylle tuotteitaan tiettyihin eheystasoihin. Turvallisuuden eheystaso kertoo miten toimintavarmasta ja/tai käyttövarmasta järjestelmästä ja toiminnosta on kyse. Tason perusteella valitaan myös tason saavuttamiseksi vaadittavat suunnittelu- ja arviointimenetelmät. Eheystason perusteella asiakas osaa päätyä luotettavuudeltaan sopivaan järjestelmään.

Turvallisuuden eheystasomenettely ohjaa suunnittelua ja helpottaa turvallisuuden arviointia. CERD-projektissa eheystasoperiaatetta kehitetään myös muille ohjelmistojen laatukriteereille kuin turvallisuuskriittisyydelle.

Tässä luvussa esitetään lyhyt katsaus standardin esittämään ajatukseen turvallisuuden eheystasoista. Myös muut standardit tai ohjeet luokittelevat turvallisuutta hieman muunnelluin periaattein. UK MoD 00-56 ja MIL-Stan 882C ovat ehkä niistä tunnetuimmat. Muita ovat lisäksi MISRA, DO 178B ja FDA (510k). Esityksen tavoitteena on taustojen hakeminen eheystasojen välittömälle hyväksikäytölle ohjelmistopohjaisten järjestelmien suunnittelussa ja arvioimisessa joko kehitystyön aikana tai jälkeen.

EN-IEC 61508:n hyödyllisyys on siinä, että se määrittää kuhunkin turvallisuuden eheystasoon (TET) tason saavuttamiseksi vaadittavat ja suositeltavat suunnittelu-, toteutus- ja arviointimenetelmät ja -tekniikat. Nimenomaan tämän ominaisuuden hyödyntämisestä on CERD-projektissa kyse. Hyödyntämiskohteina ovat toimintavarmuusattribuutti käytettävyys ja ylläpidettävyys. Tarkastellaan mahdollisuuksia kehittää vastaavanlaisia deterministisiä sääntöjä vähemmän kriittisille aloille kuin turvallisuus. Tosin menettelyn kehittäminen turvallisuusattribuutillekaan ei ole pois suljettu, sillä menettelyä on runsaasti arvosteltu. Tarkastellaan seuraavaksi EN-IEC 61508 TET-menettelyn ongelmakohtia.

Turvallisuuden eheystason menettelyn periaatteelliset hyödyt ovat ilmeiset, mutta käytännössä menettelyn soveltaminen on vaikeaa ja niistä puuttuvat uskottavat arviointikriteerit. Vaikeuksia on seuraavien toimenpiteiden soveltamisessa: 1) TLJ:n määrittämisessä oikealle TET:lle, 2) TET:n allokointi ohjelmistolle ja 3) TET:n täyttymisen osoittaminen.

4.1 Kattostandardi EN-IEC 61508

4.1.1 Turvallisuuden eheystason määrittelytapa

EN-IEC 61508 suosittaa sekä kvantitatiivisia että kvalitatiivisia määrittelymenetelmiä, joita esitellään standardin lisäksi monissa viitteissä (mm. ATU 2001).

Kvalitatiivista riskigraafia on sovellettu Suomessa kemianlaitosten ja voimalaitosten kriittisen automaation luokittelussa. Puutteena määrittelyssä on riskigraafin epälineaarisuus. Tulos selvästi johtaa joko liian alhaiseen tasoon tai liian korkeaan tasoon. EN-IEC 61508:ssa riskigraafi annetaan tässä vain esimerkkinä. Tarkistus olisi, että tietylle teollisuusalueelle perustettaisiin sopivat riskigraafit, jotka riittävän hyvin yhtenäistävät määrittelyprosessia niin, että eri sovelluksissa päästäisiin samaan tiettyyn turvallisuuden eheystasoon.

Lindsay ja McDermid (1997) ehdottavat standardien turvallisuuden eheystasojen systemaattisuutta parantavia määrittelytapoja. Heidän kritiikkinsä kohdistuu erityisesti EN-IEC 61508:aan, mutta myös muut vastaavat standardit saavat osansa. Parannusehdotukset koskevat

- vakavuustason määrittämistä
- ohjelmiston virhetoimintojen todennäköisyyksien määrittämistä
- hyötyjen ottamista huomioon
- alhaisten turvallisuuden eheystasojen mukaan ottamista
- arviointikriteerien tarkastelua.

Vakavuustasojen määrittely on heidän mukaansa useimmiten järjestelmä- ja ohjelmistokehittäjien hallinnan ulkopuolella. Tämä merkitsee eittämättä yhteistoiminnallisia vaikeuksia riittävän informaation hankkimisessa. Usein ei päästä yksimielisyyteen kaikista vaaran vakavuuteen vaikuttavista tekijöistä. Esimerkiksi teollisuuslaitosten suojausjärjestelmillä on monia järjestelmän ulkopuolisia vakavuustasoihin vaikuttavia tekijöitä. Niitä ovat vaaratilanteesta ilmoittaminen, pakoreitit, läsnäolokiellot jne. Usein nämä tekijät ovat tiedossa, mutta eheystason määrittelyn kuluessa muuttuvat, ehkä nimenomaisesti määrittelystä johtuvina. Standardeissa riskien yhteismitallisuus olisi yritysten tasapuolisen kohtelun vuoksi suotavaa. Eheystasojen määrittely on yritysten riskinoton sanelemaa.

Ohjelmiston virheet ovat systemaattisia virheitä. Siksi on hyvin vaikea määritellä vaaran esiintymistodennäköisyyttä. Pyritään suunnitteluratkaisuin alentamaan esiintymistodennäköisyyttä. Vikasietoisessa suunnittelussa tavoitteena on löytää satunnaisvikamahdolli-

suudet ja estää niiden eteneminen järjestelmätason vikaantumisiksi. Turvallisuuskriittisessä ohjelmistopohjaisten järjestelmien suunnittelussa tulisi päästä samaan. Defence-in-depth-periaatteella suunnitellaan eri suojauskerroksia järjestelmien ympärille. Periaate vaatii vikamekanismin hienoista mallintamista siten, että syyt ja seuraukset saadaan selville. Niitä tarvitaan pyrittäessä määräämään järjestelmän esiintymistodennäköisyys komponenttien luotettavuusarvioista. Vioittumismallista tulee väkisinkin hyvin kompleksinen ja siten myös kaikkea muuta kuin kustannustehokas.

EN-IEC 61508 esittää, että riskien määrittelyssä tulee myös *saavutettavat hyödyt* ottaa huomioon. Riskien vähentäminen ei siten saa olla liian kallista hyötyihin verrattuna. Erityisesti turvallisuusanalyysit keskittyvät vain vaaroihin suosittaen riskien vähentämistä ilman hyötynäkökohtien tarkastelua. Turvallisuusanalyyseista erilliset ryhmät tekevät päätöksiä riskinoton suuruudesta usein vielä eheystasojen määrittelystä erillään.

Turvallisuuden eheystasot koskevat vain korkean turvallisuuden järjestelmiä. Ohjelmistotuotanto kuitenkin kasvaa ja laajenee tuottaen asiakkaalle tärkeitä *alemman eheystason komponentteja*. Niiden ei kaikkien tarvitse liittyä turvallisuuteen, vaan toimintavarmuus ja käyttövarmuus ovat myös oleellisia tekijöitä. Myöskään tukiohjelmien, jotka kantavat vastuuta korkean luokan eheystason ohjelmiston tekemisessä, ei välttämättä tarvitse olla samaa korkeaa eheystasoa. Erityisesti tämä koskee COTS-ohjelmistokomponentteja.

Uudelleen käytettäville ja COTS-ohjelmistokomponenteille riippumattomuusvaatimus on usein sovelluskohtaisesti liian tiukka. Vaatimuksen mukaan järjestelmä perii korkeimman turvatoiminnon turvallisuuden eheystason (EN-IEC 61508), joka siihen asetetaan. Vaatimus pätee koko turvatoiminnon alueella, siis myös järjestelmän ulkoisille järjestelmille, jos niissä jotakin turvatoiminnon osaa prosessoidaan. Joissakin standardeissa, kuten DoD 00-56, taso voi vielä siitäkin kasvaa.

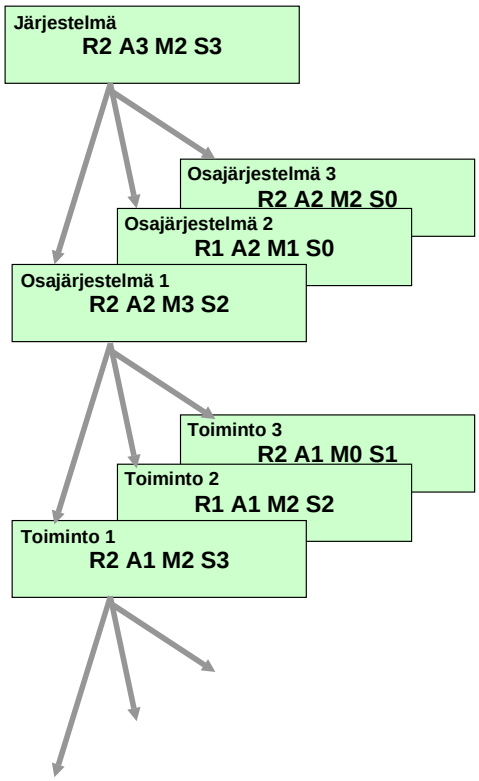
Käytännössä ostaja joutuu tekemään aikaisessa vaiheessa päätöksen turvallisuuteen liittyvän järjestelmän hankkimisesta ja ostamisesta. Usein siinä vaiheessa ei ole vielä tiedossa täsmällisesti oikea turvallisuuden eheystaso. Kun eheystaso on määritelty, voidaan päätellä osuiko ostopäätös oikeaan tasoon. Määrittelyn ja päättelyn jälkeen eheystason valintaa, arviointia tai testaamista ei useinkaan käytännössä enää tehdä, mikä saattaa johtua *eheystasojen määrittelyn arviointikriteerien* puuttumisesta. Kehitysprosessissa tulisi arviointikriteerien avulla päätellä oikeista eheystasoista sekä riittävän perusteellisista kehitysprosesseista että tuotteen tarkistuksista, eikä vain siitä noudatettiinko oikeaa prosessia.

4.1.2 Turvallisuuden eheystason allokointi ohjelmistolle

EN-IEC 61508 esittää, että allokointi TLJ:n tasolta osajärjestelmiin ja ohjelmistoille tapahtuisi todennäköisyyspohjaisesti. Periaate sopii lähinnä laitteistoille. Standardi sisälsi vielä vuosikymmen sitten esityksen allokointimenettelystä riippuville osille, mm. ohjelmistolle, mutta se liian vaikeatajuisena poistettiin. Lääkintälaiteteollisuuden standardeissa on vain yksi turvallisuustaso (merkittävä LOC), johon kriittinen ohjelmisto voi lukeutua. Ohjelmisto perii samat vakavuustasovaatimukset kuin koko tuote.

Laitteiston TET:n allokointi ei tuottanekaan ongelmia. Riippuvien virheiden tarkastelu ja huomioon ottaminen allokoinnissa on ensiarvoisen tärkeää. Systemaattisten virheiden käsittelyssä on mahdollista kokeilla luotettavuusprofiilia, joka ottaa huomioon turvallisuuden lisäksi muutkin luotettavuusattribuutit, jotka ovat toimintavarmuus, käytettävyys ja ylläpidettävyys. Kaikkien luotettavuusattribuuttien mukaan ottaminen merkitsee TET-periaatteen soveltamista myös muille kuin turvallisuudelle. Tämä taas mahdollistaa myös muiden kuin TLJ:n mukaan oton menettelytavan piiriin. Niistä ehkä keskeisimmät ovat tänä päivänä valmisohjelmistot, mm. COTSit.

Luotettavuusvaatimusten allokoinnissa voidaan hyödyntää nk. luotettavuusprofiilia (kuva 3), mikä on esitetty Tekesin ETX-tutkimusohjelman projektissa tuotetussa julkaisussa (Harju 2000). Luotettavuusprofiilin allokointi vaihe vaiheelta noudattaa luonnollisella tavalla ylhäältäalas -tekniikkaa, koska toiminnalliset ja ei-toiminnalliset vaatimukset luotettavuustoimintoja ja -vaatimuksia myöten jaetaan samalla tekniikalla samoille kohteille. Kuvan 3 esityksessä on huomattavaa, että turvallisuusattribuutti on keskitetty vain osajärjestelmälle 1, osajärjestelmiä 2 ja 3 ei ole määritelty turvallisuuskriittisiksi. Edelleen kuvasta havaitaan, että tietyn luotettavuusattribuutin taso vähenee yleensä suunnittelun kuluessa, mutta se voi myös keskittyä, jos toimintoa käytetään useammassa muussa järjestelmässä. Ohjelmistojen kohdalla COTSit ja käyttöjärjestelmät ovat useissa kohteissa käytettyjä ja siten niihin myös kohdistuvat korkeimmat luotettavuusvaatimukset.



Kuva 3. Esimerkki luotettavuusprofiilin allokoimisesta. Järjestelmän luotettavuusprofiili allokoidaan osajärjestelmille ja toiminnoille. Kuvassa R=Reliability, A=Availability, M=Maintainability, S=Safety (Harju 2000).

4.1.3 Turvallisuuden eheystason saavuttamisen osoittaminen

Ongelmana on osoittaa, että TLJ on saavuttanut tietyn turvallisuuden eheystason vaatimukset. EN-IEC 61508 suosittaa kahdenlaista vaihtoehtoista tapaa: kvantitatiivista Bayesian-osoittamistapaa etenkin kahdelle korkeimmalle turvallisuuden eheystasolle (TET 3 ja TET 4) ja determinististä tapaa kahdelle alemmalle tasolle (TET 1 ja TET 2).

Kvantitatiivinen tapa koskee lähinnä laitteistoa – ohjelmistolle ja systemaattisille virheille pätisi standardin mukaan vain hyviksi todetut suunnittelu-, analyysi- ja testausmenetelmät, ns. deterministinen tapa. Niitä standardissa onkin suositettu runsaasti. Ei ole tiedossa yhtään tapausta, missä suositusten toimivuus käytännön projekteissa olisi onnistunut ja olisi selvästi pystytty osoittamaan tietyn menetelmäjoukon validius. Tämä johtuu ensisijassa systemaattisen tarkastelutavan puutteesta. Ei ole myöskään osoitettu, ettei menettelytapa toimisi.

Luotettavuuden osoittamistavoissa EN-IEC 61508:n yhtenä heikkoutena on standardin keskittyminen prosesseihin tuotteen sijasta (ks. kappale Kolmannen osapuolen serti-

fioinnit luvussa 6) Etenkin ydinvoimalaitosalalta on tullut viestejä siitä, että laadukas toteuttaminen ei yksin riitä, vaan todisteiden pitäisi kohdistua ensisijaisesti itse tuotteeseen. COTS-ohjelmistoista ei yleensä ole edes saatavilla kehitysprosessin laatutodisteita.

4.2 Militääristandardit MoD 00-55/56

Militääristandardi (MoD 00-56 1996) määrittelee turvallisuusohjelman elektroniikkaa sisältäville järjestelmille. Vaatimukset kattavat turvallisuuden hallinnan, analyysimenetelmät ja verifiointitoimenpiteet. Standardi soveltuu kaikkiin projektin elinkaarivaiheisiin määrittelystä aina järjestelmän poistamiseen. Ohjelmiston turvallisuuden eheystasomäärittelyt tarvittavine turvallisuusmenetelmineen sisältyvät standardiin.

Toinen militääristandardi (MoD 00-55 1997) keskittyy yksin ohjelmistoon eritellen vaatimuksia turvallisuuteen liittyvälle ohjelmistokehitykselle. MoD 00-56 koski järjestelmän turvallisuutta. Vaatimukset kattavat määrittelyn, suunnittelun, koodauksen, testauksen ja integroinnin. Turvallisuuden hallinnan vaatimuksiin kuuluvat turvallisuussuunnittelu, -analyysi, -katselmukset, -auditoinnit ja ohjelmiston turvallisuuden koetarkastelun (safety case) läpivienti. Standardi suosittaa työvälineitä ja menetelmiä sekä ohjelmiston kehittämiselle että testaamiselle.

Turvallisuuden eheystaso on todennäköisyysmitta sille, että järjestelmä täyttää turvallisuusvaatimukset. Tasoja on neljä, joista korkein S4 kohdistuu kaikkein kriittisimmille järjestelmille ja matalin S1 vähimmin kriittisille. Määrittelyt ovat hyvin samankaltaiset kuin standardilla EN-IEC 61508. Tässäkin turvallisuuden eheys määritellään turvatoiminnoittain, jotka järjestelmään asennetaan. Järjestelmä omaa sen kriittisimmän toiminnon turvallisuuden eheystason. MoD 00-56 ohjeistaa turvallisuuden eheystason jakamisen turvatoiminnon toteuttaville komponenteille. Useat standardien vaatimuksista on lievennetty vähiten kriittisille komponenteille.

Turvallisuuden eheystason määrittäminen tapahtuu seuraavasti:

1. Arvioidaan kunkin mahdollisen onnettomuuden vakavuus. Tasoja on neljä katastrofaalisesta merkityksettömään.
2. Määrätään korkein siedettävä todennäköisyys kullekin onnettomuudelle.
3. Määrätään tavoitearvo jokaisen vaaran todennäköisyydelle ottamalla huomioon vaarasta onnettomuuteen johtava tapahtumaketju.
4. Määrätään turvallisuuden eheystaso.

Turvallisuuden eheystason määrittämisessä standardi MoD 00-55 ottaa huomioon EN-IEC 61508:sta poiketen funktioiden lukumäärän, kuten taulukoista 1 ja 2 ilmenee. Turvallisuuden eheystason allokoitumisesta tai periytymisestä standardi antaa samat ohjeet kuin EN-IEC 61508 joskus 90-luvun alussa. Ohjeet poistettiin, koska niitä ei ymmärretty.

Taulukko 1. Turvallisuuden eheystason määrittäminen ensimmäiselle tai ainoalle toiminnolle.

Onnettomuuden vakavuus			
Katastrofaalinen	Kriittinen	Vähäinen	Mitätön
Taso S4	Taso S3	Taso S2	Taso S1

Taulukko 2. Turvallisuuden eheystason määrittäminen toiselle ja sitä seuraaville toiminnoille.

Ensimmäisen toiminnon vikatiheys	**Onnettomuuden vakavuus**			
	Katastrofaalinen	Kriittinen	Vähäinen	Mitätön
Tiheästi toistuva	Taso S4			
Mahdollinen		Taso S3		
Satunnainen			Taso S2	
Vähäinen				
Epätodennäköinen			Taso S1	

MoD 00-55 sisältää vaatimukset vain ohjelmiston turvallisuuden eheystasolle neljä (S4). Standardi suosittaa kaikille ohjelmiston kehitysvaiheille tietynlaisia verifiointi- ja validointiratkaisuja, jotka eivät oleellisesti poikkea EN-IEC 61508:n linjasta. Ratkaisut ovat ensisijassa formaaleita menetelmiä, jotka ovat myös IEC:n standardin valikoimissa. Katselmuksia vaaditaan mm. verifiointien varmistamiseksi, mitä ei EN-IEC 61508:ssa vaadita suoraan, mutta varmistamistavasta on kerrottava turvallisuuden hallinnan toimenpiteissä.

4.3 Militääristandardi MIL-STD 882C

Militääristandardi (MIL-STD 882C 1993) muistuttaa kohdassa 4.2 esitettyjä standardeita, mutta eroaa näistä monessakin suhteessa. Standardin sovellusalana ovat kaikki turvallisuuteen liittyvät järjestelmät, eivät pelkästään ohjelmoidut. Turvallisuuden eheystasot määritellään sekä vaaran vakavuustasojen että esiintymistodennäköisyyden tasojen avulla. Vaaran vakavuustasoja on neljä katastrofaalisesta merkityksettömään ja vaaran esiintymistodennäköisyyden tasoja viisi tiheästi toistuvasta epätodennäköiseen.

Standardi kuvaa kriittisen ohjelmiston riskin arviointiin soveliaita menetelmiä. Koska ohjelmistojen virhetoimintojen todennäköisyyttä on vaikea laskea, standardi vaihtaa tämän käsitteen ”ohjelmiston valvontakategoriaksi”. Valvontakategorioita on neljä:

I Ohjelmisto valvoo automaattisesti laitteiston vikaantumista siten, että valvontatehtävän epäonnistuminen johtaa vaaratilanteeseen.

II Ohjelmisto valvoo laitteiston vikaantumista, mutta valvontatehtävän epäonnistuessa joko riippumaton turvallisuusjärjestelmä tai operaattori vastaanotettuaan informaatiota ohjelmistolta huolehtii turvallisuudesta.

III Ohjelmisto jakaa käskyjä kriittiselle laitteistolle, mutta valvonta vaatii tarvittaessa operaattorin väliintulon ja kutakin varaa suojataan redundanttisilla turvallisuustoimenpiteillä. Tai ohjelmisto laatii ja viestii informaation turvallisuudesta ja redundanttinen turvallisuustoimenpide suojaa vaaroilta.

IV Ohjelmisto ei valvo turvallisuuskriittistä laitteistoa eikä viestitä turvallisuuskriittisistä asioista.

Vakavuustasoista ja valvontakategorioista voidaan muodostaa riskimatriiseja. Tästä standardi antaa esimerkin, jossa ohjelmisto luokitellaan viiteen tasoon. Korkein riski on valvontakategorioilla I ja II sekä katastrofaalisella vakavuustasolla. Analyysit ja testit tulisi tehdä riskitason mukaisesti, mutta tarkempaa yksilöintiä ei anneta.

Oheinen ohjelmiston valvontakategoria on vaikea tulkita, eikä se edes ole kattava. Kategoriaa koskevia ohjeita on hyvin niukasti. Myös riskin määrittäminen kuvatulla tavalla sopii vain tiettyyn sovellusympäristöön. Siinähän suositetaan, että riskiä ohjelmistovirheen johtamiseen onnettomuuteen vähennetään antamalla lisää ohjaksia operaattorille. Tapa voi sopia joihinkin militäärisovelluksiin, mutta tuskin on yleistettävissä mm. tapauksiin, joissa vasteajat ovat hyvin lyhyitä tai joissa operaattori saattaa olla ylikuormitettu.

4.4 Ilmailualan standardi DO-178B

Siviili-ilmailualan standardi (DO-178B 1992) tukee ohjelmiston sertifiointia, ja sitä vaati eräässä VTT Automaation ohjelmiston luotettavuuden analysointiprojektissa myös ESA[5].

Standardi määrittelee viisi vakavuustasoa, jotka ovat hyvin spesifisiä ilmailualalle: A) vikaantuminen estää turvallisen lentämisen ja laskeutumisen, B) miehistö ei kykene havainnoimaan tiettyjä tiloja, C) miehistön havainnointimahdollisuudet ovat rajoittuneet, D) miehistön työkuorma on kasvanut ja E) vialla ei ole vaikutusta lentokoneen toimintaan tai miehistön työkuormitukseen. Ohjelmisto luokitellaan näihin luokkiin, jos sen virhetoiminnosta on seurauksena jokin näistä ilmiöistä. Esiintymistiheyttä standardi ei luokittele.

Standardi suosittaa menetelmiä yleisesti ohjelmiston kehittämiselle, vaikka ohjelmiston elinkaarivaiheita siinä ei ole määritelty, ja vakavuustasoille suositukset koskevat vain testaamista. Testaamisen tavoitteita on kaksi: 1) on demonstroitava, että ohjelmisto täyttää vaatimukset, ja 2) demonstroitava, että ohjelmistovirheet, jotka voivat johtaa vaarallisiin virhetoimintoihin, on poistettu. Testaamiselle on vakavuustasokohtaisia suosituksia.

4.5 Rautatiestandardit EN 50126 ja EN 50128

Rautateitä koskeva standardi EN 50126 (1999) käsittelee myös muita luotettavuusattribuutteja kuin turvallisuus. Siinä kuvataan luotettavuuden hallintaprosessi, luotettavuusvaatimusten määrittelyprosessi ja prosessi, jossa luotettavuusvaatimusten täyttyminen demonstroidaan. Luotettavuuden tavoitearvoja standardi ei kuitenkaan anna, eikä se keskity yksinomaan ohjelmistoihin, vaan mukana ovat ohjelmistoa sisältämättömätkin järjestelmät.

Standardin EN 50128 sovellusalana ovat ohjelmistoja sisältävät ohjaus- ja suojausjärjestelmät. Sovellusala koskee sekä turvallisuuteen liittyvää että liittymätöntä ohjelmistoa, joita ovat sovellusohjelmistot, käyttöjärjestelmät, tukityövälineet, kiinteät ohjelmistot ja COTSit.

EN 50126 määrittelee neljä turvallisuuden eheystasoa, joihin pääsee riskin arvioinnilla. Riskin arviointiprosessi kuvataan standardissa, mutta siinä ei anneta ohjeita riskin luokittamiseen. Turvallisuuden eheys vastaa vikaantumistodennäköisyyttä.

---

[5] European Space Agency

Turvallisuuden eheystasojen allokoinnille standardi antaa yksiselitteisen ohjeen: järjestelmän TET allokoidaan osajärjestelmälle siten, että osajärjestelmä omaa siihen kohdistetun korkeimman vaatimuksen TETin. TETin määräytymisessä otetaan huomioon mahdollinen osajärjestelmän kahdentaminen. Jos osajärjestelmä sisältää ohjelmiston, sen TET vastaa osajärjestelmän TETiä.

TET-kohtaiset menetelmät suunnittelulle ja verifioinnille annetaan standardissa EN 50128. Standardi sisältää myös TET 0:n siten, että myös turvallisuuteen liittymättömälle ohjelmistolle voidaan suosittaa menetelmiä. TET-kohtaisten menetelmien suosittamistapa muistuttaa kattostandardin EN-IEC 61508 suosittamistapaa.

4.6 Lääkintälaitestandardit ja -ohjeet

Monet muut standardit ja ohjeet määrittävät turvallisuuteen liittyvän järjestelmän suoraan tietylle turvallisuuden eheystasolle[6]. FDA (1998) luokittelee ohjelmistoja sisältävät lääkintälaitteet vikaantumisten turvallisuusvaikutusten perusteella kolmeen vakavuustasoon, joista kuitenkin vain yksi on varteenotettava vaatimustenmukaisuuden osoittamisessa. Luokitustavassa ohjelmiston virhetoiminnan tai jonkin piilevän virheen seuraukset voivat olla joko välittömiä, eli suoraan lääkintälaitteesta tai sen ohjelmistosta aiheutuvia vaikutuksia, tai välillisiä, mm. muista hoitotoimenpiteistä johtuvia.

Lääkintälaitteen vakavuustaso on *merkittävä* (Major LOC[7]), jos se voi aiheuttaa potilaan, käyttäjän ja/tai asianomaisen sivullisen henkilön kuoleman tai vakavan loukkaantumisen; *kohtuullinen* (Moderate LOC), jos laite voi aiheuttaa lievän loukkaantumisen; *vähäinen* (Minor LOC), jos ei ole odotettavissa, että aiheutettaisiin edes lievää loukkaantumista.

FDA on ohjeistanut merkittävään vakavuustasoon tiettyjä sääntöjä ohjelmiston virhetoiminnan ja piilevän virheen esiintymistodennäköisyyden alentamiseksi. Säännöt ovat suunnittelua ohjeistavia, mutta myös analysoinneille ja testauksille annetaan ohjeita.

Yksityiskohtaiset menetelmäsuositukset kuitenkin puuttuvat lääkintälaitestandardeista ja -ohjeista, mutta FDA julkaisee raportteja havaituista vioista. Wallace ja Kuhn (2000) esittävät tutkimusraportissaan myös suosituksia menetelmiksi ohjelmistovirheiden havaitsemiseen ja ennaltaehkäisemiseen. Heidän suosittamistapaansa käsitellään tämän julkaisun kohdassa 4.6. Käsittelyn keskeisin tulos on siinä, että menetelmävalinnat ovat hyvin riippuvia virhetyypeistä ja virhetyyppien luokittelu on monimutkaista.

---

[6] Termit niissä ovat toiset.
[7] Level of Concern

EU:n lääkintälaitedirektiivi ohjaa yksityiskohdat ohjelmistoa sisältävän tuotteen suunnittelulle standardiin (IEC601-1-4 1996), joka soveltuu kriittisille laitteille eikä siten sisällä määriteltyjä luokituksia. EU:n alueella lääkintälaitteiden valmistajat noudattavat yleisesti myös FDA:n ohjeita, ja siten kriittinen ohjelmistoa sisältävä tuote kuuluu vakavuustasoltaan merkittävään luokkaan.

4.7 Eräs tutkimus menetelmien suosittamisesta

Mitä luotettavuuden arviointimenetelmiä todella tarvitaan? Sopivien menetelmien valinnassa tulisi ottaa huomioon luotettavuuden vaatimustasot, ohjelmiston ja järjestelmän monimutkaisuus, suunnittelutavat jne. Tarkastellaan seuraavaksi lähteen (Wallace & Kuhn 2000) esittämää tapaa arvioida menetelmätarvetta lääkintälaitteiden ohjelmistojen laadunvarmistuksessa.

FDA ylläpitää tietokantaa lääkintälaitteista löytyneistä vioista, jotka ovat tulleet ilmi joko järjestelmätesteissä, asennuksen aikana tai potilaskäytössä. Wallace ja Kuhn (2000) kävivät läpi vuosina 1983–1997 esiin tulleet viat, joiden alkusyy oli ohjelmistossa (383 kappaletta).

Tekijöiden tarkastelutavassa lähtökohtana ovat havainnot järjestelmän käyttäytymisestä, mm. tiedon häviäminen tai korruptoituminen, tarvittavan hälytyksen puuttuminen, järjestelmän odottamaton pysähtyminen tai näyttöinformaation ristiriitaisuus eri näyttölaitteiden välillä. Vian ilmitulotavat jaettiin kolmeentoista luokkaan (taulukko 3).

Taulukko 3. Vikojen ilmitulotavat lääkintälaitteissa (Wallace & Kuhn 2000).

Vian ilmitulotapa	Osuus vioista
Toiminto (function) yhdessä moduulissa: tavallisesti laskutoimitus	29 %
Väärä siirto tai liike (behavior): esim. nosturilaite	22 %
Ulostulo (output): yleensä jonkun toiminnon tulos seuraavalle toiminnolle	19 %
Puutteellinen palvelu (service): koostuu useista palveluista useilla moduuleilla	10 %
Näyttövirhe (display): kuvat, luvut, teksti eri formaateissa	8 %
Sisäänmeno (input): alustussisäänmeno, tietokanta, tiedosto	4 %
Virheellinen vaste (response): toiminnon odottamattomuus	3 %
Data: virheellinen tieto tai tiedon katoaminen	1 %
Laatuvirhe: tuote ei vastannut laatuvaatimuksia	1 %
Järjestelmävirhe (system)	1 %
Ajastusvirhe (timing) instrumenteilla tai laitteen palveluilla	1 %
Ohjeistusvirhe (user instruction)	1 %
Yleisvirhe (general): virhettä ei voida luokittaa riittävän tiedon puuttuessa	0 %

Vian todennäköisin syy ilmoitettiin kolmessatoista luokassa (taulukko 5). Vikaluokittelua varten he tutustuivat useaan systeemiin, joista Beizer (1990) on julkaissut yhden tunnetuimman. Vikaluokitteluun hyväksyttiin vain 342 vikaa, koska osasta ilmoitettuja vikoja ei ollut luokitteluun tarvittavia tietoja. Luokittelua hankaloitti vielä se, että monet virhetoimintojen syyt olisivat kuuluneet useampaan kuin yhteen luokkaan. Joissakin tapauksissa ei selvinnyt olisivatko ohjelmistokehittäjät jättäneet tietyn vaatimuksen kokonaan määrittelemättä, vai olisiko vika johtunut virheellisestä ohjelmalogiikasta. Jos edellinen tapaus olisi ollut totta, vika olisi luokiteltu vaatimusviaksi, joita ovat laiminlyönti, ristiriitaisuus, epäselvyys, moniselitteisyys jne. Jälkimmäisessä tapauksessa oikea luokittelu olisi ollut logiikkavirhe. Tutkijat tyytyivät suosittelemaan tällaisessa tapauksessa hyviä määrittelymenetelmiä ja sijoittivat vian logiikkavirheeksi. Täsmällinen luokittelu olisi vaatinut tarkempia tietoja vioittumistapahtumista kuin mitä heillä oli saatavilla.

Taulukko 4. Vian todennäköisimmät syyt lääkintälaitteissa (Wallace & Kuhn 2000).

Vikaluokka	Osuus vioista
Logiikka (logic): logiikkavirhe vaatimusmäärittelyssä, yksi tai useampi odottamaton samanaikainen tila, väärät raja-arvot	43 %
Laskenta (calculation)	24 %
Muutoksen vaikutus (change impact)	6 %
Data: väärät yksiköt, arvot, ongelmat todellisissa sisäänmenoarvoissa	5 %
Virheellinen tai puuttuva vaatimus (requirements):	4 %
Toiminnon puuttuminen (omission)	3 %
Muu virhe (other): COTS, EPROM, kovo, muistit, vaatimusten toteuttaminen	3 %
Laadunvarmistus (quality assurance): riittämätön prosessi, validoinnin puuttuminen uudessa versiossa	3 %
Ajastus (timing)	3 %
Alustus (initialization)	2 %
Liityntävirhe (interface)	2 %
Rakenteenhallinta (configuration management)	1 %
Vikasietoisuus	1 %

Ehkä em. perusteluin ylivoimaisesti suurin osa raportoiduista vioista johtuikin ohjelmiston loogisista virheistä (43 %). Seuraavaksi suurin osuus johtui vääristä laskentatuloksista (24 %). Alun perin luokkia oli 28, joista tekijät joutuivat supistamaan niitä kolmeentoista, mm. laskentavirheissä oli alun perin useampia alaluokkia. Hierarkkinen tapa esittää vikoja olisi ollut kuitenkin kuvaavampi tapa, mutta luokittelutapaa voidaan supistaa, jos luokittelun tavoite eli sopivien menetelmien suosittaminen luonnistuisi.

Tekijät analysoivat, millä toimenpiteillä viat olisi voitu joko ehkäistä tai havaita ennen laitteen luovutusta asiakkaalle. Liitteen A taulukossa A3 esitetään esimerkkejä tekijöiden suosittamista ehkäisevistä ja havainnoivista menetelmistä.

Toimenpidesuositukset on annettu esimerkkeinä, sillä kovin yleistä suositusmallistoa ei tutkijoiden mukaan ollut mahdollista laatia. Toimenpidesuositukset on annettu vikaluokittain ja lisäksi ongelmatapauksittain. Viitatussa julkaisussa he esittävät malliksi vain yhden ongelmatapauksen vikaluokkaa kohti. Toimenpiteet he jakavat ehkäiseviin ja

havaitseviin. Ehkäisemisellä he tarkoittavat toimenpiteitä, joihin olisi kehitysprojektissa pitänyt ryhtyä ennen testiä ja havaitsemisella menetelmiä testausvaiheen ja laadunvarmistusvaiheen aikana. On mahdollista, että suositeltavia toimenpiteitä olisi tehtykin, mutta ilmeisesti jokin niissä ei ollut onnistunut. Joka tapauksessa suositukset perustuvat hyvään käytäntöön, mikä on esitetty viitteessä (Paulk et al. 1993) laatujärjestelmänä Capability Maturity Model, sekä viitteissä (Wallace & Ippolito 1994, Wallace et al. 1996).

Tärkeimpien virheiden eli logiikkavirheiden lähtökohtana voivat olla kehitysvaiheet (määrittely, suunnittelu ja koodaus) sekä ylläpito. Tutkijat suosittavat ennaltaehkäisemiseen sekä vaatimusten jäljitettävyysanalyysia koko kehitysprosessille määrittelystä koodaukseen että koodin vertaamista suunnitteluun. Havaitsemisessa tulisi tarkistaa, katselmoida ja testata koodit. EN-IEC 61508 (2001: osa 3) suosittaa joukkoa staattisia analyyseja (so. tarkistuksia ja katselmointeja), mm. raja-arvoanalyysit, tarkistuslistat, ohjaus- ja tietovuoanalyysit. Menetelmien valinta riippuu tarkasteltavasta tapauksesta, lähinnä TETistä ja mahdollisista virhetyypeistä.

Tekijät suosittavat yhteenvetoartikkelissa yhteensä n. 50 kappaletta sekä ennaltaehkäiseviin että havainnoiviin menetelmiin. Suositukset ovat laajalta alalta sekä yksityiskohtaisia tiettyjen elinkaaren vikoihin puuttuvia että yleisluontoisia päivittäiseen hyvään laatukäytäntöön puuttuvia menetelmiä (mm. tuotteen hallinta, muutosten vaikutusanalyysi, formaalit menetelmät, simulointi, määrittelyn, suunnittelun ja koodin katselmukset, lukuisat erilaiset testit, jäljitettävyys ja jäljitettävyysanalyysi). Yksityiskohtaiset suositukset ovat ensi sijassa tarkistusluontoisia menetelmiä (riittääkö vikasietoisuus, datan arvoalueen ja validisuuden tarkistus).

Yleisluontoiset suositukset kattavat EN-IEC 61508:n menetelmäsuositukset suhteellisen tarkkaan. Yksityiskohtaisista, vikoihin kohdistuvista suosituksista saa sen käsityksen, että yleisluontoisiin menetelmiin tulisikin suosittaa tarkistuslistoja toimialakohtaisesti.

5. Valmisohjelmistojen ja niistä koostuvien sovellusten luotettavuus

Tulevaisuudenkuvaksi ennustetaan minkä hyvänsä uuden järjestelmän kokoamista olemassa olevasta koodista. Onpa jopa esitetty tämän väitteen koskevan konservatiivista turvallisuuskriittisten ohjelmistojenkin tuotantoa (Voas 1998a, IAEA 1999). Tietoturvan merkitys valmisohjelmistojen hyödyntämisessä on selvästi normaalia ohjelmistokehitystä tärkeämpää (Lindqvist & Jonsson 1998).

Seuraavassa kohdassa tarkastellaan COTS-ohjelmistoihin liittyviä käsitteitä ja ongelmia, kohdassa 5.2 kahta amerikkalaista valmisohjelmistojen ja niistä koostuvien järjestelmien kelpoistusmallia sekä lopuksi kohdassa 5.3 luotettavuusnäkökulmia suunnitteluun ja arviointiin.

5.1 COTS-ohjelmistomäärittelyitä

5.1.1 Mitä ovat COTS-ohjelmistot?

COTS-käsitteelle annetut määritelmät vaihtelevat jonkin verran. Yleisesti ottaen termillä tarkoitetaan vapaasti myynnissä olevaa tuotetta, jonka ostaja voi ottaa osaksi omaa järjestelmäänsä. Siten COTS tarkoittaa mitä hyvänsä valmisohjelmistoa esimerkiksi yrityksen oman toiminnan sisällä.

COTS-ohjelmiston koko voi vaihdella hyvin pienestä ohjelmistokomponentista suureen ja monimutkaiseen ohjelmistotuotteeseen. Pieni ohjelmisto voi koostua toimintolohkokirjaston komponenteista, jotka on konfiguroitu ja parametroitu tietyn toiminnan toteuttamiseksi. Näkökulmasta riippuen sellaiset tietokonejärjestelmien perusohjelmat, kuten käyttöjärjestelmät, laiteajurit, kääntäjät, linkkerit yms. lasketaan COTSeiksi.

Ydinvoima-alalla tunnistetaan seuraavanlaisia COTSeja (IAEA 1999):

1. Olemassa olevat laiteperheet, jotka on kvalifioitu ydinvoimaloiden sovellukseksi. Järjestelmäohjelmisto, tukiohjelmisto ja sovellusohjelman automaattinen koodigeneraattori.
2. Olemassa olevat laiteperheet, jotka on kehitetty sovellukseksi yleisesti teollisuudelle. Järjestelmäohjelmisto, tukiohjelmisto ja sovellusohjelman automaattinen koodigeneraattori. Niissä on mm. laitoksen automaation diagnostiikkaa ja ylläpidettävyydestä huolehtivia ominaisuuksia, kenttäväyläsovittimia ja käyttöliittymiä.

3. Olemassa oleva ohjelmisto yleisiin sovelluksiin. Kääntäjät, tietopankit, standardi simulointiohjelma.
4. “Black box” -laitteiden johonkin tiettyyn tehtävään tarkoitettu sulautettu ohjelmisto, SMART-lähetin, kommunikointimoduuli, liitäntäohjelma tietoliikenneväylien välillä.

Yleiskäyttöisten PC-koneiden ja niiden käyttöjärjestelmien yleistyessä erilaisissa laiteohjaustehtävissä on COTS-ohjelmiin kohdistuva kiinnostus kasvanut. Esimerkiksi tietokantasovellusten ja graafisten käyttöliittymien ohjelmoiminen alusta asti itse on monen erikoisalan ohjelmistotuottajan kannalta epätaloudellista, kun kaupan hyllyltä löytyy useita valmiita ratkaisuja, jotka vaativat parhaimmillaan vain hieman konfigurointia toimiakseen. Laitevalmistajan näkökulmasta COTSeista onkin hyötyä silloin, kun ohjelmistosta pystytään löytämään suunnitteluvaiheessa sopiva lohko, joka voidaan toteuttaa COTS-ohjelmalla. Kuva luetteloi muutamia selkeimpiä COTS-ohjelmistokomponenttien etuja ja haittoja.

COTS-hyödyt

- saatavilla välittömästi
- monipuoliset, valmiit toiminnot
- valmistaja vastaa kehitys- ja ylläpitokustannuksista
- toiminnot valmiiksi testattuja
- laajasti käytetty, hyvät luotettavuustiedot
- tiheä tuoteparantelu
- laitteisto/ohjelmisto-riippumattomuus
- teknologiatrendien nopea noudattaminen
- hyvät tulevaisuuden kuvat: täydellinen vaihtokelpoisuus ja uudelleenkäyttö
- sovelluksen luotettavuus saataa rakentua epäluotettavistakin tuotteista

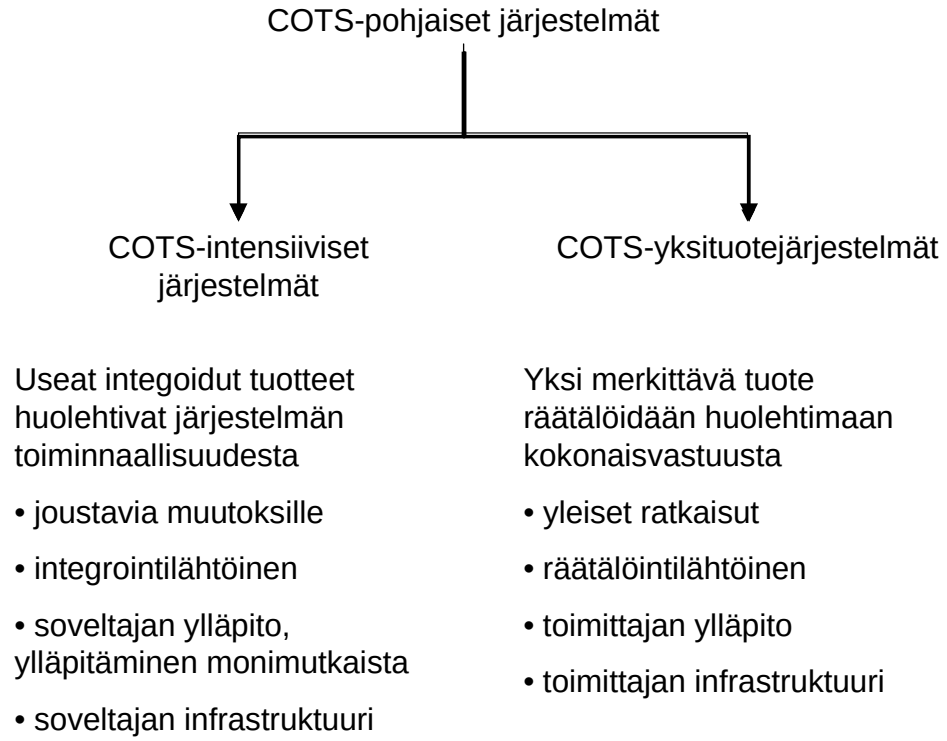
COTS-haitat

- lisensiointi- ja yksilöllistämisviiveet
- kalliit lisensiointimaksut
- toistuvat ylläpitomaksut
- lähdekielinen koodi ei käytettävissä, huonot luotettavuuden kontrollointimahdollisuudet
- luotettavuus usein tuntematon tai riittämätön
- ostajalle ylimääräiset ominaisuudet uhkana käytettävyydelle ja suorituskyvylle
- valmistajan päivityksiin ja ylläpitoihin ei kontrollointimahdollisuutta
- integrointi hankalaa, eri valmistajien tuotteet yhteensovittamattomia
- useiden valmistajatuotteiden synkronointivaikeudet

Kuva 4. COTS-ohjelmistokomponenttien hyödyt ja haitat.

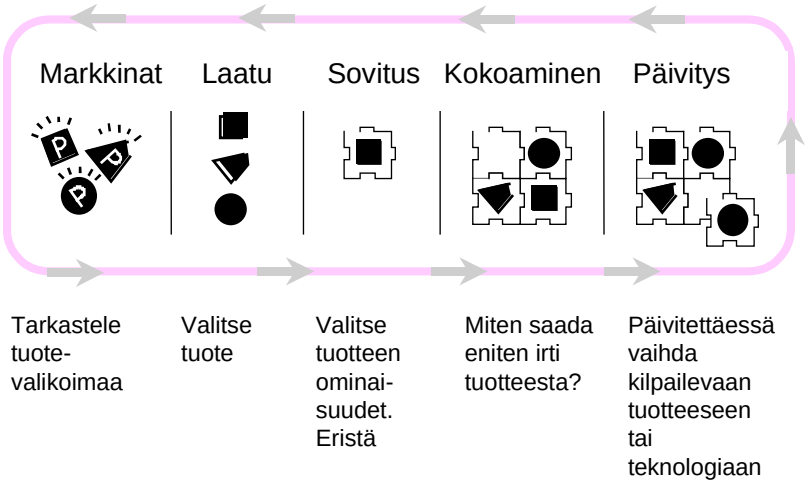
5.1.2 COTS-järjestelmät

COTS-pohjaiset järjestelmät voidaan määritellä ja jakaa usealla tavalla, raportissa Wallnau et al. (2001) järjestelmät jaetaan kuvan esittämällä tavalla intensiivisiin eli monituotejärjestelmiin ja yksituotejärjestelmiin.



Kuva 5. COTS-pohjaiset järjestelmät voidaan jakaa yhden tai useamman komponentin hyödyntämisen pohjalta. Wallnau et al. (2001).

Yksituotejärjestelmät koostuvat pääasiassa yhden merkittävän komponentin ympärille. Ne ovat paljon yksinkertaisempia ja helpommin evaluoitavia kuin jälkimmäiset, jotka koostuvat useasta järjestelmän toimintoja toteuttavasta COTS-komponentista. Komponenttien lukumäärä ei kuitenkaan ole erottava tekijä, vaan monimutkaisuus. Yhden tuotteen ympärille toteutettu järjestelmä voi olla myös kompleksinen, esimerkiksi tietokantaohjelman päälle on rakennettu merkittävästi lisää toiminnallisuutta. Toisenkaltainen monimutkaisuus ilmenee, kun järjestelmä kootaan useasta erilaisesta ohjelmistokomponentista, jotka joko ovat kaupallisesti saatavia tai kototekoisia tai molempia.



Kuva 6. Yksinkertaistettu malli COTS-intensiivisten järjestelmien elinkaarelle (Wallnau et al. 2001).

COTS-tuotteen tai -tuotteista koostuvan järjestelmän evaluointia esittää yläpuolinen kuva. Kuva esittää yksinkertaistettua COTS-komponenteista koottujen intensiivisten järjestelmien elinkaarta. Siihen kuuluvat:

- tuotetarjonnan arviointi
- kelvollisten tuotteiden valitseminen
- sopivien tuoteominaisuuksien valitseminen
- järjestelmän kokoaminen
- järjestelmän päivittäminen.

Evaluointi kuuluu oleellisena osana jokaiseen kuvan esittämän elinkaaren vaiheeseen. Valitettavaa on kuitenkin, ettei kirjallisuudesta erityisemmin löydy evaluointikäytäntöä tukevaa tekstiä. Miten esimerkiksi rajata tuotevalikoiman tarkastelua? Pitäisikö käydä kaikki tuotteet läpi vai onko jokin metodi läpikäynnin rajoittamiseksi. Mistä tiedetään, että meillä on oikea lista käytössä, sillä markkinoilla saattaa olla runsaasti tuotteita saatavilla? Merkittävää kuitenkin on, että COTS-ohjelmistokomponentin ja siitä koostettavan järjestelmän evaluointi on tehtävä asteittain kaikissa kuvan esittämissä vaiheissa jättämättä mitään evaluointiosuutta pois.

Kehittämistarpeita määrittelevässä tutkimuksessa (Niemelä et al. 2000) todettiin, ettei kotimaisilla yrityksillä ole käytettävissään komponenttien hankintaprosesseja. Käytän-

nöt ongelmat tulivat esille tutkimuksessa suoritetuissa haastatteluissa. Seuraavassa lyhyt yhteenveto ilmenneistä ongelmista ja haasteista:

- Kilpailussa mukana pysyminen vaatii standardienmukaisuutta, mutta standardit, joita kaupalliset komponentit noudattavat, laajenevat ja elävät voimakkaasti. Siten komponenttien kyky tarjota standardiyhteensopivuutta vaihtelee.

- Heterogeeniset komponentit lisäävät ylläpitokustannuksia ja komponenttien yhteensopivuusriskiä.

- Erityisesti kaupallisten komponenttien istuttaminen sulautettuun ympäristöön tuottaa hankaluuksia. Komponentit vaativat sovitusliitännän.

- Komponenttien integroitavuutta ei voida todeta etukäteen, mikä merkitsee kustannustehokkuuden alenemista. Testausvaiheesta voi tulla pitkä ennen lopullisen hankintapäätöksen tekemistä.

- Pitkäaikaisen toiminnallisuuden toteaminen onnistuu vain rasitustesteillä suunnitellussa käyttöympäristössä. Suoritusympäristön rakentaminen aiheuttaa ylimääräisiä investointikustannuksia.

Tutkimus toteaa lisäksi, että tarvitaan uusia elinkaarimalleja komponenttien valitsemiseksi ja integroitavuuden tueksi. Malleilla kyettäisiin korjaamaan yhteensopimattomuutta, joustamattomuutta, monimutkaisuutta ja muuttuvuutta. Pienet komponentit otetaan käyttöön nopeasti valinnan jälkeen, mutta isot vaativat erillisen evaluoinnin.

Kuten edellä todettiin, kaupallisten komponenttien evaluoinnista ei ole kehitetty kunnollista menettelyä. Puutteen takia komponenttien valmistajat eivät ota huomioon evaluointia ohjelmiston kehittämisprosessissa, vaan asiat käsitellään tapauskohtaisesti arvioimalla uusien versioiden vaikutukset vanhan version käytölle. Evaluointi jää komponentin käyttäjän tai integraattorin vastuulle.

5.1.3 Komponenttipohjainen ohjelmistokehitys

Komponenttipohjaisella ohjelmistokehityksellä (component based software development, CBSD) tarkoitetaan sekä suunnittelutekniikoita että työkaluja, joiden avulla sovelluksia kootaan uudelleenkäytettävistä valmisohjelmista tai ohjelmistokomponenteista laajamittaisen ohjelmoinnin sijasta. Sillä tavoitellaan lisää tehokkuutta ja joustavuutta ohjelmistokehitykseen. CBSD:llä tavoitellaan seuraavia ominaisuuksia:

- Kaupalliseen termiin liittyy oleellisena osana se, että valmistaja pyrkii tekemään tuotteestaan mahdollisimman yleiskäyttöisen, toisin sanoen saamaan samalla tuotantokustannuksella laaja-alaisesti asiakkaita
- Yleiskäyttöisyys ja laaja-alaisuus lisää käytettävyyttä, luotettavuustietoja saatavilla
- ”Ennalta valmis” merkitsee myös sitä, että asiakas voi hyödyntää pilottitarkasteluita

Komponenteista kootaan sovelluksia, joissa komponentit kommunikoivat rajapintojensa kautta. Kytkentöjä rakennetaan esim. graafisella työkalulla tai kevyitä makroja (scripts) käyttäen. Olemassa oleva komponentti tai perinteinen ohjelma voidaan myös ”kuorruttaa” siten, että sen palveluita rajataan tai tiedoille tehdään tarvittavia muunnoksia. Komponenttien kokoamisen pohjana voidaan käyttää sovelluskehystä. Se on periaatteellinen arkkitehtuuri, joka määrittelee, miten komponentit toimivat yhdessä tietyllä sovellusalueella. Esimerkiksi. VTT:llä tällaista on kehitetty panosprosessien ohjaukseen (Kuikka 1999).

Komponenttipohjaista ohjelmistokehitystä on kahta päätyyppiä:

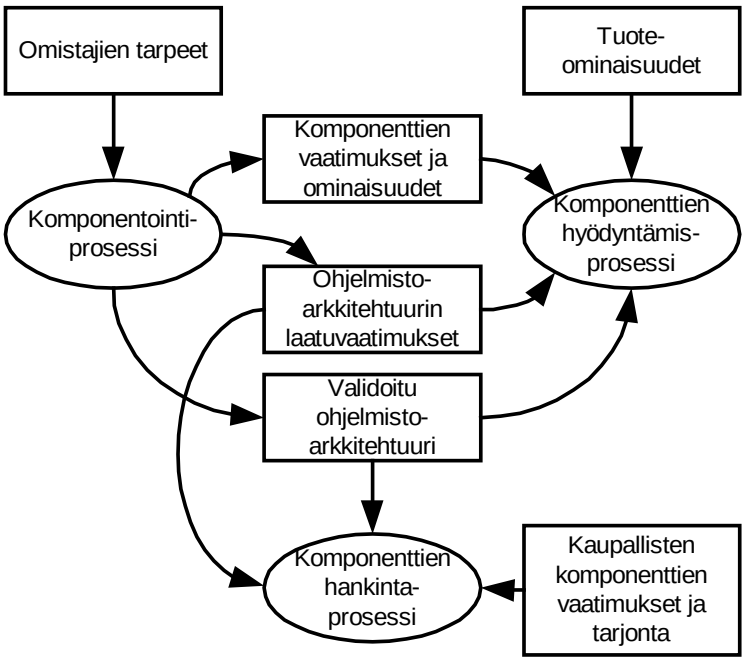
1. Tuotteen tai sovelluksen kokoaminen markkinoilta hankittavista ohjelmistokomponenteista (Commercial Off-The-Self, COTS).
2. Ohjelmistokomponenttien tuottaminen ja kehitysprosessin muuntaminen komponentteihin perustuvaksi, mikä voidaan rinnastaa yrityksen sisäiseksi (tai alihankinta) tuotteistamiseksi ohjelmistotuotannossa.

Arkkitehtuurien tai sovelluskehysten kehittäminen ei kuitenkaan ole ainut kehityssuunta. Nykyään tutkitaan myös erilaisia rajapintatekniikoita, ja skaala ulottuu yksinkertaisesta syntaktisesta muuttujien ja funktioiden yhteensopivuudesta monenlaisiin dynaamisesti neuvotteleviin rajapintoihin (mm. agenttitekniikat).

Valmisohjelmien käyttö siirtää ohjelmistokehityksen painopistettä prosessielinkaaren alkuun päin. Toisaalta silloin ladataan myös suuria paineita ohjelmistopalasten integroitavuuteen ja ennen kaikkea komponenttien laatuun.

Tekesin ETX-teknologiaohjelmassa tehtiin äsken kansallinen strateginen selvitys Teolliset komponenttipohjaiset ohjelmistot, Kehittämistarpeet ja toimenpide-ehdotukset (Niemelä et al. 2000). Selvityksen mukaan ohjelmistokomponenttien käyttö kasvaa, mutta itse komponenttien kehittämisen pääongelmina nostettiin esiin rajapintojen määrittelyn vaikeus ja testauksen hankaluus. Hyödyntämisongelmista suurimpia olivat kaupallisten komponenttien laadun vaihtelu, haluttujen ominaisuuksien puuttuminen ja do-

kumentoinnin määrällinen ja laadullinen riittämättömyys. Muualla maailmassa tehdyt vastaavantapaiset selvitykset osoittavat samaa.



Kuva 7. Komponenttipohjaisen ohjelmistokehityksen liiketoimintaprosesseja.

Komponenttipohjaisuus muuttaa ohjelmistokehitysprosessia. Kuva on mukailtu em. teknologiakatsauksesta. Menestyksellinen toiminta edellyttää, että toimintatavat ja -prosessit ovat kunnossa kautta linjan. Avainasemassa ovat tuotekehityksen organisointi, ammattitaitoinen henkilöstö, kokenut arkkitehtuurisuunnittelu, hyvät työkalut, toimivat ohjelmistomarkkinat ja yritysverkostot.

Laajan ohjelmiston kehittäminen on työlästä, mutta vielä työläämpää se on silloin, kun myös sitä pyörittävä laitteisto ja koko järjestelmä on suuri. Kehitystyökalut ovat kehittyneet ja mahdollistaneet yhä suuremmat sovellukset. Toisaalta kiristyvä kilpailu ja markkinatarpeet vaativat yhä tehokkaampia ominaisuuksia ja yhä enemmän toimintoja ohjelmistoihin. Ohjelmistoprojektien koot kasvavat vuosi vuodelta. Tarkastellaan tässä kappaleessa miten luotettavuusnäkökulmalla kyetään parantamaan ohjelmiston kehittämistä ja luotettavuuden osoittamista projektipäällikölle, julkistajalle, asiakkaalle ja kolmannelle osapuolelle.

Jaaksi et al. (1999) esittävät erään perättäiskehittämisen metodiikan laajamittaiselle ohjelmiston rakentamiselle. Perättäiskehittämisessä tai evolutiivisessa kehittämisessä lisätään uutta toiminnallisuutta modifioimalla vanhaa. Perättäiskehittämisen pääominaisuus on vuosien inkrementaalinen kehittäminen. Täydentävää kehittämistä patistavat markkinat, eikä perättäiskehitettävää järjestelmää voida pitää täydellisenä vielä kun markkinatarpeita on jäljellä. Yleensä lisäykset ja parantelut ovat ylläpitoa, mutta kehitystyössä

tulisi jo ottaa huomioon jatkuvan olemassa olevan ohjelmiston pohjalta tapahtuva kehittäminen.

Laajan järjestelmän luotettavuuteen vaikuttavat seuraavat neljä tekijää:

1. monimutkaisuus
2. testien kattavuus
3. ylläpidettävyys
4. kehittäjän taito.

Nämä tekijät korreloivat toistensa kanssa. Monimutkaisuus lisää testaustarvetta ja ylläpidettävyyttä sekä monimutkainen suunnittelukohde vaatii myös taitavan suunnittelijan. Jos testauskattavuutta vähennetään, voi ylläpidettävyyden tarve kasvaa.

5.1.4 Ohjelmistotekniikan ongelmat

Laajan ohjelmiston keskeiset ongelmakohdat ovat ohjelmiston konfiguraation hallinnassa, projektin hallinnassa, ohjelmiston ylläpidossa ja integroinnissa (Jaaksi et al. 1999). Konfiguraation hyvää hallintaa varten tulisi olla kohteita, jotka kyetään tunnistamaan ja jakamaan rakenteellisesti. Komponentit ovat konfiguraation hallinnan näkökulmasta tärkeimmät avainelementit, sillä esimerkiksi perättäiskehittämisessä sovellukset rakennetaan komponenteista, sovellustuotteet sovelluksista ja lopulta järjestelmätuotteet sovellustuotteista.

Tiimeille ja yksilöille tulisi jakaa sopivan kokoisia moduuleita. Saisi olla vain jokunen riippuvuus osajärjestelmien välillä. Tällä tavoin vähennetään virhealttiutta, kun jollakin osaprojektilla on ongelmia osajärjestelmänsä kanssa. Lisäksi toiminnallisuuden ja laadun arviointi ja seuranta helpottuvat.

Virheen korjauksen jälkeen tulisi tarkastella mahdollisia korjauksesta aiheutuneita negatiivisia vaikutuksia. Ylläpito helpottuu kun osajärjestelmät ovat selkeästi määriteltyjä ja höllästi kytkettyjä. Kielteiset vaikutukset ovat muutenkin epätodennäköisiä, jos virhettä joudutaan etsimään vain osajärjestelmän sisältä. Minimaaliset ja hyvin määritellyt osajärjestelmien riippuvuudet ovat edellytyksiä kustannustehokkaalle ja nopealle integroinnille.

COTSeista kootun pitkäikäiseksi tarkoitetun järjestelmän ylläpidettävyys ja hallittavuus on työlästä. Siihen kuuluvat mm. komponenttien päivittäminen, konfiguraation hallinta, vianetsintä jne. Edellisessä kohdassa esiteltiin kirjallisuuslähteitä, joissa selostettiin ylläpidettävyyttä ja hallintaa arkkitehtuurin näkökulmasta. Tässä tavoitteena on esitellä

kirjallisuusviitteitä, joissa suositellaan tiettyjä menetelmiä ylläpidettävyyteen ja haalintaan.

Vigder (1998) rakentaa puitteet tietynlaiselle tarkastuslistakäytännölle, missä selvitetään, onko COTS-pohjaisella järjestelmällä tietyt toivotut arkkitehtuuriominaisuudet. Tarkistuslista sopii sekä viranomaisen että asiakkaan (käyttäjän) käsiin (Vigderin työ kohdistuu militäärialalle). Tarkistuslistamenettely ohjaa myös arkkitehtuurin rakentamista.

Julkaisu myös listaa lukuisia tarkistuskohtia, joista monet koskettavat luotettavuuteen liittyviä kysymyksiä. Niistä saadaan käsitys 1) siitä miten monipuolisista asioita tulee ottaa komponenttipohjaisen suunnittelussa ottaa huomioon, ja 2) soveltuuko tarkistuslistamenettely COTS-pohjaisten järjestelmien luotettavuuden arviointiin tai osaksi arviointijärjestelmää. Liitteessä A on taulukkoon A4 poimittu muutamia luotettavuuteen liittyviä osia Vigderin tarkistuslistoista.

Yleiset COTSien evaluointiin sopivat aiheet eivät aina sovellu COTS-komponenteista kootuille järjestelmille. Esimerkkinä näistä aiheista ovat seuraavat:

- evaluointi on kertaluontoista, siinä useita tuotteita verrataan yhteiseen joukkoon (painotettuja) kriteereitä
- evaluointi on luonteeltaan hyväksymistestin kaltaista toimintaa, siinä asetetaan vaatimukset ja vaatimuksenmukaisuus tuotetta kohden evaluoidaan
- evaluointi on riippumatonta toimintaa kehitysprosessista
- evaluoinnille voidaan soveltaa geneeristä mallia.

COTS-komponenteista koottujen järjestelmien evaluoinnin kaksi ääripäätä olisivat

1. määritellä kaikki ominaisuudet kaikille kyseeseen kuvitelluille tuotteille, joita mahdollisesti valita järjestelmään, ja
2. kokeilla tuotteita käyttämällä niitä niin kauan kuin kyetään.

Viimeksi mainittu vaihtoehto on yleinen lähinnä riskittömyytensä johdosta.

5.1.5 Laajat markkinat – hyvät luotettavuustiedot, vajaat tuotetiedot

Yksi COTSsien eduista on laajat markkinat, joista saa hyvän luotettavuuskuvan mikäli ohjelmistokoodia ei muuteta. Laajat markkinat takaavat alhaiset kehityskustannukset, mutta muutokset ja uudet ohjelmaversiot pilaavat laajasta käytöstä kertyneet luotetta-

vuustiedot. Jos koodi on räätälöity yksittäiseen käyttöön, kyse ei ole enää COTS-ohjelmistosta, vaan normaalista ohjelmistosta normaalein laadun ja luotettavuuden osoitustarpein. Etenkin suuret toimittajat ovat usein innottomia myymään tuotteita, joiden menekki on vähäistä, tai jos se olisikin suurta, niin ne eivät mielellään toimita tuotteidensa laatu- ja luotettavuustietoja.

Digitaaliteknologian nopea uusiutuminen johtaa muutostarpeisiin ja vanhentuneisuuteen. Tarve vanhojen osien uusimiselle on välttämättömämpää kuin analogisilla laitteilla. Muutosten hallinnastakin on usein vaikea saada tietoa, mikä johtuu monien saatavilla olevien COTS-tuotteiden patenttioikeuksista. Valmistajat saattavat tehdä sisäisiä tuotemuutoksia ilmoittamatta siitä käyttäjille. He olettavat, että jos tuote on edelleen sopiva ja sillä on oikeat toiminnot ja ominaisuudet, muutoksista ei tarvitse ilmoittaa. Monilla aloilla kuitenkin on tiukat tuotteen- ja versiohallinnan vaatimukset, eikä kyseisen kaltainen muutostoiminta ole ongelmatonta hyväksyntäprosesseissa. Ongelmia aiheutuu myös yritettäessä säilyttää analogiset järjestelmät, sillä niiden tai niiden varaosien saatavuus voi olla heikkoa.

Saatavuudessa COTSit, erityisesti ohjelmistot, menevätkin selvästi analogisten vastaavuuksiensa edelle. COTSeista koostuvan järjestelmän kehittäminen on nopeaa, sillä suunnittelussa käytettävät kohteet ovat jo olemassa ja usein saatavilla tietoverkon välityksellä. Jakelu on nopeaa ja saatavuus yleensä maailmanlaajuista. Maailmanlaajuisuus viittaa laajaan ja monipuoliseen käyttökokemustietoon, mikä edesauttaa kypsien ja korkealaatuisten tuotteiden kehittämistä sekä edistää suunnittelijan luottamusta tuotteeseen. Lisäksi varaosien saatavuus on hyvä, koska laaja käyttäjäkunta myös vaatii COTS-komponenttien varaosia käyttöönsä. Tosin komponenttivalmistaja saattaa kadota tai lopettaa ylläpidon, eikä korvaavaa komponenttia löydy.

Standardit COTS-ohjelmistot soveltuvat monenlaisille alustoille mukauttaen järjestelmäsuunnittelua. Mukautuvuudesta on myös hyötyä korvattaessa osia. Komponenttipohjaisuus kärsii kuitenkin vielä uuden teknologian ongelmista, työkalut ovat hankalakäyttöisiä ja nopeasti kehittyviä. Tekniikkaa ei ehkä ole riittävästi omaksuttu. Komponenttien ylläpidossa saatetaan kuunnella yhden valmistajan kehitystoiveita enemmän kuin toisen, mikä johtaa yhteensopivuusongelmiin.

Tavallisesti ostajalla ei ole minkäänlaista hallintaa COTSin valmistajan ohjelmistontuotantoprosessiin eikä oikeutta saada yksityiskohtaisia dokumentteja COTSin tuotannon eri vaiheista. Lähdekielistä koodia hän ei myöskään saa käyttöönsä. Dokumentaation puuttuminen johtaa vaikeuksiin, koska monissa tapauksissa tarvitaan tietoa luotettavuuden ohjaukseen. Ostajalla ei useinkaan ole käytettävissä tietoa valmistajan käyttämistä ohjelmointistandardeista ja verifiointimenetelmistä ja -tuloksista. Lähdekielisen koodin tarkistaminen ja white box -testit saattavat jäädä tekemättä, mikä vaikuttaa yk-

sikkö-, integraatio- ja hyväksyntätesteihin. Dokumentteja joko ei ole saatavilla mistään hinnasta tai hyvin korkeasta hinnasta. Dokumenttien puuttuminen muuttaa komponenteista koostuvan järjestelmän kehittämistä, testaamista ja ylläpitoa. Järjestelmä voi hyödyntää vain osaa COTSin toiminnallisuudesta, osan jäädessä käyttämättömäksi. Tulisiko testata vain hyödynnetty osuus? Entä mitä tehdä muulle osuudelle?

COTS-tuotteiden monikäyttöisyydestä johtuvat ylimääräiset toiminnot voivat aiheuttaa odottamattomia ja testaamattomia toimintoja. Tahattomat toiminnot ovat tulosta monimutkaisesta suunnittelusta ja jäävät tuotteeseen koska niitä ei ole huomattu ottaa testauksiin mukaan. Tuotteen monimutkaisuuden kasvattaminen johtuu siitä, että COTSien on oltava ulkoisesti hölliä. Niillä on valmistajan mielestä kyettävä helposti toteuttamaan uusia toimintoja.

Joskus ohjelmistoartefaktien saatavuudella ei ole merkitystä. Niiden suuruus saattaa sekä kustannus- että aikatekijät huomioon ottaen hankaloittaa täydellistä verifiointia ja validoinnin tekemistä. Käytännölliseen ratkaisuun päästään sopivan tasoisilla menettelyillä, jotka saadaan esimerkiksi riskianalyysien tuloksista. Erityisesti kustannustehokkuus saavutetaan kohdentamalla riskianalyyseja tarkoitetulle käytölle. Verifiointeihin ja validointeihin sekä erityisesti riskianalyyseihin tulisi olla käytettävissä informaatiota COTS-ohjelmiston toiminnoista. Sitä saadaan käsikirjoista ja koulutuspaketeista, joista joissakin tapauksissa selviää kuinka kukin komponentti toimii siten, että riittävä toiminnallinen vaatimusmäärittely olisi konstruoitavissa.

5.1.6 Integroituvuus

Useiden ohjelmistomoduulien, erityisesti COTSien yhteen liittäminen voi aiheuttaa yllättäviä toimintoja, koska liittämisessä ja integrointitestauksissa ei ole huomattu tarkastaa moduulin kaikkia mahdollisia ominaisuuksia. Moduuliin on unohtunut ominaisuuksia. Moduuliin on jäänyt koodia, joka on ollut tarpeeton uudessa käytössä ja unohtunut jostakin syystä poistaa.

COTSin hankintahetken tulisi tavallisesti ajoittua varsin aikaiseen vaiheeseen ohjelmistotuotannossa, sillä ohjelman käytön oppimiselle, testaamiselle ja integroimiselle muuhun järjestelmään on varattava riittävästi aikaa. Niinpä hankintahetkellä ei useinkaan voida määrätä COTSin dynaamisia ominaisuuksia kovinkaan tarkasti, sillä muu systeemi ei ole riittävän valmis. On siis olemassa riski, että vaikka ohjelma hankintahetkellä täyttäisikin kaikki staattiset vaatimukset, integraatiovaiheessa ohjelma saattaa osoittautua epätarkoituksenmukaiseksi. Erityisesti aikakriittisissä sovelluksissa tämä voi olla ongelma, varsinkin jos COTSin sisäistä rakennetta tunnetaan niin huonosti, että suoritusaika-arvioita on vaikea tehdä (kasvaako lineaarisesti, neliöllisesti jne.)

Ostaja ei voi myöskään olla varma siitä, että kaikki COTSin tarvitsemat resurssit ja riippuvuudet tunnetaan. Esimerkiksi Windows-ympäristössä tämä voi johtaa vakaviin ongelmiin seuraavasti: oletetaan, että COTS käyttää jotain DLL-tiedostoa (dynamic linking library), jota ostaja ei tunne. Ohjelmisto toimii aluksi oikein. Uuden Windows-version, huoltopäivityksen tai jonkun muun ohjelman asennuksen yhteydessä DLL muuttuu. Tämän seurauksena myös COTSin toiminta muuttuu käyttäjälle yllättävällä tavalla.

Mikäli ohjelmaan integroidaan useita eri COTS-ohjelmia, on hyvin todennäköistä, että näillä kaikilla on omat tietorakenteensa. Rakenteiden kirjavuus voi hankaloittaa suunnittelua ja toteutusta. Yksinkertaisena esimerkkinä voitaisiin ajatella kahta matriisien käsittelyyn tarkoitettua ohjelmaa. Ensimmäinen ohjelma aloittaa rivien ja sarakkeiden numeroimisen nollasta, toinen puolestaan ykkösestä. Ei ole mahdotonta kuvitella tilannetta, jossa ohjelmoija indeksoi väärin, koska sotkee kutsuttavat ohjelmat.

5.1.7 Käyttöönotto ja käyttö

Koska COTSin valmistaja pyrkii saavuttamaan tuotteellaan mahdollisimman suuren asiakaskunnan, on siinä todennäköisesti ostajan haluamien ominaisuuksien lisäksi myös ylimääräisiä ominaisuuksia. Viime kädessä tämä johtaa omaa ohjelmaa suurempaan konfigurointitarpeeseen. Tällöin väärän konfiguroinnin mahdollisuus kasvaa. Väärästä konfiguroinnista voi seurata ohjelman virheellinen tai puutteellinen toiminta.

Väärä konfigurointi voi tapahtua jo tuotantovaiheessa, asennettaessa järjestelmää asiakkaalle tai käytön aikana. Asennuksen jälkeen riskin muodostavat uusien ohjelmien asennukset, jotka saattavat automaattisesti muuttaa konfigurointia tai saattavat toimiakseen vaatia käyttäjältä muutoksia. Käyttäjä ei välttämättä ymmärrä tekemiensä muutosten liittyvän mitenkään alkuperäiseen ohjelmaan. Tällaisia ristiriitoja saattaisi syntyä esimerkiksi tietokannan asetusten kanssa.

5.2 COTS-ohjelmistotuotteiden kelpoistamismalleja

Korkeissa turvallisuuden luokissa arviointi vaatii yksityiskohtaiset ohjelmistoartefaktit, joita säätelevät sovellusalan määräykset ja standardit. Tarkastellaan tässä kohdassa kolmea amerikkalaista kelpoistusmallia: ydinvoimaloiden COTS-pohjaisten tuotteiden kelpoistusmenetelmää sekä FDA:n ohjeita toimialansa valmisohjelmistojen kehittämistä, koostamista järjestelmiin sekä arviointia.

5.2.1 Ydinvoimalaitosten automaatiojärjestelmien valmistuotteiden kelpoistamismalli

Periaatteena ydinvoimalaitosten automaatiojärjestelmien valmisohjelmistojen kelpoistamisessa on noudatettava samoja ydinvoimalaitoksia koskevia suunnittelu- ja kelpoistusprosesseja kuin kehitettävän ohjelmiston kelpoistamisessa (IAEA 2001). Periaatteen mukaan samansuuruinen turvallisuustaso tulisi saavuttaa ja saavuttaminen osoittaa määräysten ja standardien hyväksymillä menetelmillä, oli järjestelmä tai sen ohjelmisto ydinvoimalaitosta varten erityisesti valmistettu ohjelmisto tai kaupallisesti, laajoille markkinoille valmistettu ohjelmisto.

Koska useinkaan valmisohjelmistoa ei ole kehitetty ja kelpoistettu ydinvoimala-alalla hyväksyttyjen menettelytapojen mukaisesti, tarvitaan korvaavia menetelmiä riittävän turvallisuuden osoittamiseksi. Menetelmät ovat yleensä analyyseja ja testejä, jotka kohdistuvat mm. ohjelmiston rakenteeseen sekä mukaan otettaviin ja pois jätettäviin toimintoihin. Turvallisuusmerkityksen edellyttämillä menettelyillä tulee dokumentoinnin olla riittävän kattava ohjelmiston versioiden hallitsemiseksi sekä muutossuunnittelun mahdollistamiseksi.

Tässä kohdassa kuvataan lyhyesti yksi vaihtoehtoinen menettelytapa, jolla kelpoistaa valmisohjelmistoja Yhdysvaltojen ydinvoimaloiden automaatiomarkkinoille. Menettelyä noudattamalla kyetään osoittamaan saman turvallisuustason saavuttaminen kuin noudattamalla ydinvoimala-alan hyväksyttyjä kehitys- ja kelpoistusmenetelmiä. Menettelytapa soveltuu sekä turvallisuudeltaan merkittävien järjestelmien ohjelmistoille että järjestelmien ohjelmistoille, joilla ei ole turvallisuusmerkitystä. Yksittäisen kaupallisen komponentin luotettavuus on todennäköisesti korkeampi kuin räätälöidyn tuotteen, vaikka tämän kehitysympäristö olisikin korkeaa laatua. Kyse onkin siitä, miten luotettavuus pystytään osoittamaan.

Periaatteessa arvioinnissa ei ole eroa, käytetäänkö olemassa olevia riippumattomia komponentteja vai laitoskohtaisia räätälöityjä komponentteja. Kummatkin täyttävät samat järjestelmävaatimukset. Komponenteista on kuitenkin välttämätöntä erottaa mustalaatikon komponentit, joiden sisäinen rakenne on saatavilla, valkolaatikon komponenteista, joiden sisäinen rakenne jää tuntemattomaksi. Koska jälkimmäisen komponenttityypin vaatimustenmukaisuutta ei voida täysin osoittaa, tulee siihen kohdentaa erityisvaatimuksia. Periaatteena on, että osoitettavuustaso on kummallakin komponenttityypillä sama.

5.2.1.1 Yleismalli

Ennen kuin ohjelmistopohjainen COTS-tuotteen sisältävä automaatiojärjestelmä voidaan ottaa käyttöön ydinvoimalaitossovelluksissa, järjestelmä on kelpoistettava ja hyväksyttävä. Viitteessä (IAEA 2001) kuvatun kelpoistus- ja hyväksyntämallin ylätasoon kuuluvat seuraavat vaiheet:

1. Valitun COTS-tuotteen kelpoistaminen
2. Elikaariprosessien määrääminen
3. Laitossovelluksen kelpoistaminen.

Laitossovelluksissa käytettäväksi tarkoitettu COTS-tuote kelpoistetaan, kelpoistusprosessit, ylläpitoprosessit ym. elinkaariprosessit valitaan sekä COTS-tuote tai siitä koostuva järjestelmä kelpoistetaan ydinvoimalaitossovelluksiin käyväksi järjestelmäksi. Yleismallin ensimmäinen vaihe alkaa siitä, kun COTS-tuote on valittu.

COTSille on määritelty joukko ominaisuuksia, jotka sen on täytettävä. Tästä vaatimusmäärittelystä ilmenevät esimerkiksi COTSin syöttö- ja ulostulorajapinnoille sekä toiminnallisuudelle asetetut ehdot. Valittavan COTS-ohjelman perusteellinen testaaminen ei useinkaan ole käytännössä mahdollista ohjelmien monimutkaisuuden vuoksi. Kuten edellä mainittiin, on ulkopuolisen ohjelmistotuottajan prosessista usein mahdoton saada tietoa ja testiraporttienkin saaminen voi olla hyvin vaikeaa.

Ei-halutut toiminnot voivat tavanomaisten ohjelmointivirheiden lisäksi johtua valmistajan tahallisesti asettamista salaovista. Salaovi aktivoidaan sen tekijän tuntemalla herätteellä, jolloin tapahtuu käyttäjän kannalta jotain odottamatonta ja usein haitallista. Salaovien asettaminen ei ole kaupallisen ohjelmistotuottajan edun mukaista, joten tällaisten todennäköisyys ei vaikuta kovin suurelta. Koska COTS on ajettava ohjelma, sen mukana voi tulla yksi tai useampia tietokoneviruksia. Näiltä voidaan kuitenkin suojautua melko tehokkaasti perinteisin virustorjuntakeinoin.

Hankittavan COTS-ohjelman virheellisten toimintojen todennäköisimmät syyt ovat ohjelmointivirheissä. Kuten jo yllä todettiin, ohjelman täydellinen testaaminen on ostajalle mahdotonta. Sitä se on myös valmistajalle. Niinpä useimmat valmistajat pyrkivät ohjaamaan testausresurssejaan mahdollisimman tehokkaalla tavalla. Tämä voidaan toteuttaa esimerkiksi käyttöprofiilien avulla. Tällöin testejä pyritään tilastollisessa mielessä painottamaan käyttäjien oletettua käyttötapaa vastaavaksi. Mikään ei kuitenkaan takaa sitä, että COTSin valmistajan laatimat profiilit vastaisivat tietyn ostajan tapausta.

Toisaalta, jos sama COTS-ohjelma on ollut todistettavasti pitkään menestyksellisesti käytössä vastaavissa olosuhteissa kuin mihin sitä ollaan hankkimassa, sen luotettavuudelle voidaan katsoa kertyneen kokeellista näyttöä, jollaista on vaikea itse tuotetulle ohjelmalle nopeasti saada ja jonka merkitystä ei tule ylenkatsoa.

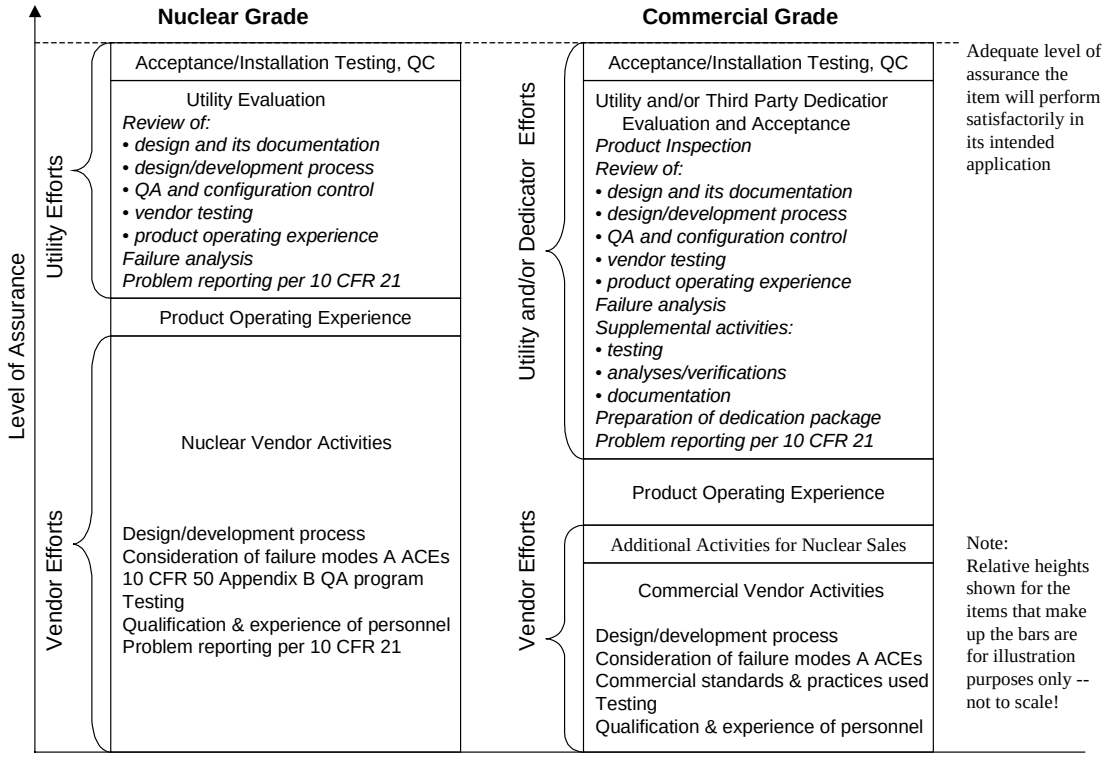
Ensimmäisen vaiheen kelpoistamisprosessi on vastaavanlainen kuin muutkin kelpoistamisprosessit ydinvoimalasovelluksissa: ympäristötestaukset, EMI/RFI -testaukset, ohjelmiston arvioinnit, järjestelmätestaukset jne. Kelpoistamisen voi tehdä toimittaja, käyttäjä (hyödyntäjä) tai jokin kolmas osapuoli. Kelpoistamistoimenpiteiden jälkeen päätetään tuotteen hyväksyttävyydestä ydinvoimalaitossovelluksiin.

Kelpoistamismallin (IAEA 2001) mukaan positiivisen hyväksyntäpäätöksen jälkeen tulee määrätä prosessit turvallisuudelle merkittävien COTS-pohjaisten järjestelmien kelpoistamisen ylläpitämiseksi, tuotteen hallitsemiseksi ja virhetietojen keräämiseksi. Prosessien määräämisellä halutaan tukea mahdollista järjestelmän myöhempää muuttamista ja helpottaa uudelleenkelpoistamista. Organisaation on myös määritettävä vastuut kelpoistuksen ylläpidolle, tuotteenhallinnalle ja virhetietojen keruulle. Turvallisuudelle merkityksettömissä sovelluksissa hyödyntäjä voi vapaasti valita mitä yllä mainittuja prosesseista määrittää, jos määrittää mitään.

Kolmannessa vaiheessa kelpoistettu tuote kelpoistetaan laitossovellukseen, mikä sisältää tuotteen integroimisen järjestelmäksi. Ydinvoimalasovelluksissa perinteiset kelpoistusmallit koskevat laitteistoa, elektroniikkaa ja mekaniikkaa. Mallit on kuvattu raporteissa EPRI (1988) ja EPRI (1994) yksityiskohtaisesti. Ne eivät koske ohjelmistopohjaisia järjestelmiä, mutta täydennettyinä ovat sovellettavissa myös COTS-ohjelmistotuotteista koostuville järjestelmille.

5.2.1.2 Kaupallisen ohjelmistoista koostuvan valmistuotteen kelpoistaminen

Kaupallisia, laajoille markkinoille tarkoitettuja tuotteita, jotka tähtäävät myös ydinvoimalasovelluksiin Yhdysvaltojen ydinvoimala-alalla (CFR 1995), koskevat samat säännöt kuin eksplisiittisesti ydinvoimaloihin tarkoitetuilla tuotteilla. Sääntöihin kuuluvat valmistusprosessien ja laadunvarmistuksen vaatimukset, joihin siinä tapauksessa, että tuote ei ole kehitetty ja kelpoistettu ydinvoimalasovelluksien sääntöjen mukaisesti, vaaditaan kuvan 8 esittämällä tavalla lisätodisteita, jotka voivat painottua mm. käyttökokemustietoihin, testeihin ja analyyseihin.



Kuva 8. Vakuuttumisen vastaavuus eksplisiittisesti ydinvoimalasovelluksiin tarkoitettujen ja kaupallisesti saatavien digitaalisten järjestelmien välillä (IAEA 2001).

Kuvassa 8 verrataan kaupallisesti saatavan digitaalisen valmistuotteen kelpoistamista järjestelmään, joka on eksplisiittisesti kelpoistettu ydinvoimasovelluksiin. Kuvan pylväissä esitettyjen toimenpiteiden laajuudet ovat sattumanvaraisia, ne esitetään vain vakuuttautumisen havainnollistamiseksi. Käytännössä toimenpiteiden suhteellinen osuus voi vaihdella huomattavasti riippuen erityissovelluksesta, toimittajasta ja arvioitavasta tuotteesta.

Kuvan vasen pylväs keskittyy erityisesti ydinvoimatarpeisiin kehitettyihin tuotteisiin. Vakuuttautumisen pääpaino on toimittajan puolella, jolla on oltava käytössään määräyksen 10 CFR 50 Appendix B (CFR 1975) edellyttämät laadunvarmuustoimet. Toimittajan menettelyt eivät yksin riitä, vaan kuten kuvan vasen pylväs osoittaa, tarvitaan lisäksi käyttökokemustietoja tuotteesta vastaavissa olosuhteissa sekä käyttäjän toimenpiteitä vakuuttauduttaessa soveltuvuuskelpoisuudesta.

Käyttäjän toimenpiteistä merkittävimmät ovat toimittajan toimenpiteiden katselmointi ja hyväksyntätestit. Edellinen sisältää tuotteen suunnittelun, toimittajan käyttämän kehitysprosessin ja laadunvarmistuksen katselmoinnit, mikä ohjelmistopohjaisille järjestel-

mille tarkoittaa ohjelmiston tuotteenhallinnan, todennuksen ja validoinnin sekä testauksen katselmointia.

Tarvittaessa haetaan apua standardeista, joista amerikkalaisille sopivat erityisesti IEEE (1993) ja ANSI/ISA (1996), mutta myös monet muut soveliaat ohjelmistotuotannon standardit ja ohjeet. Ohjetta EPRI (1993) käytetään suunnattaessa lisensiointia ohjelmistopohjaisten järjestelmien erityisaiheisiin. Vika-analyyseilla sekä tunnistetaan järjestelmän merkittävät vioittumistavat että tarkastellaan toimittajan prosesseja mahdollisten vioittumistapojen ja poikkeavien tilanteiden tunnistamiseksi. Jos järjestelmä on ollut käytössä, katselmoidaan käyttökokemukset tyydyttävän toiminnan toteamiseksi. Käyttäjä voi myös katselmoida toimittajan suunnittelu- ja laadunvarmistuskäytännöt. Kun järjestelmä on saapunut, käyttäjä tekee vastaanottotarkastukset ja hyväksyntätestit omilla menettelytavoillaan.

Kuten yllä olevan kuvan esityksestä havaitaan, vakuuttautumisessa oletetaan, että kaupallisesta tuotteesta ei ole saatavissa vastaavaa määrää informaatiota kuin nimenomaan ydinvoimalasovelluksiin tarkoitetulla digitaalisella tuotteella (vasen pylväs). Koska toimittajalla ei ole käytössään määräyksen 10 CFR 50 Appendix B (CFR 1975) edellyttämiä laadunvarmuustoimia, toimittajan osuutta joudutaan sopivassa määrin kompensoimaan lisätoimenpiteillä, joita ovat testaus, analysointi ja dokumentointi. Kompensointimäärä riippuu mm. siitä, mitä poikkeamia toimittajan laadunvarmistusjärjestelmällä on Appendix B:ssä kuvattuun vastaavaan järjestelmään.

Laajassa käytössä olevan tuotteen hyvillä käyttökokemustiedoilla saattaa olla ratkaiseva merkitys tätä kelpoistusmallia noudatettaessa. Kokemuksia on muualta kuin ydinvoimasovelluksista, mutta tiedon määrä saattaa olla huomattavan paljon suurempi kuin mitä kapealta ydinvoimasektorilta on mahdollistakaan saada. Tiedon täytyy kuitenkin soveltua suunniteltuun ydinvoiman sovellusympäristöön ja olla riittävää sekä yksikkömäärän, toiminta-ajan että hyvyyden osalta.

Mahdollisesti käyttäjän on hankittava lisätodisteita kaupallisen järjestelmän riittävästä turvallisuudesta. Niitä voivat olla mm. seuraavat toimenpiteet:

- lisätestit järjestelmän oikean toiminnallisuuden tai tiettyjen mahdollisten poikkeamien toteamiseksi
- lisäkatselmukset ja -analyysit riittävän suunnittelun toteamiseksi ja mahdollisten merkittävien vioittumistapojen tunnistamiseksi
- lisädokumentaatio tiettyjen puutteellisesti dokumentoitujen prosessien riittävyyden toteamiseksi.

On selvää, että nämä esitetyt lisätoimenpiteet eivät lisää tuotteen laatua tai turvallisuutta, vaan auttavat ja dokumentoivat vakuuttauduttaessa riittävästä laadusta ja turvallisuudesta.

Kaupallisesti saatavien digitaalisten tuotteiden kohdalla erityisesti käyttäjä joutuu tekemään enemmän laadun ja turvallisuuden osoittamistehtäviä kuin käyttäjä vastaavissa suoraan ydinvoimasovelluksiin tähtäävien tuotteiden osalta. Lisätehtävien määrään vaikuttavat monet seikat, mm. :

- ensisijaisesti turvallisuusmerkitys ja taloudellinen riski
- kuinka täsmällisesti vaatimustenmukaisia ovat toimittajan kehitys- ja laadunvarmistusprosessit olleet
- kaupallisesti saatavan järjestelmän valmiusaste
- järjestelmän monimutkaisuus, mitä monimutkaisempi sitä laajemmat varmistautumistoimet.

Käyttäjän täytyy tapaus tapaukselta arvioida miten paljon työtä kyetään taloudellisten rajojen puitteissa toimittajan prosessien ja dokumentoinnin päälle tekemään. Myös on päätettävä vaihtoehtoisten tuotteiden välillä, panostaako halvempaan tuotteeseen ja kalliimpaan arviointiprosessiin kuin kalliiseen, mutta vähän maksavaan arviointiprosessiin.

5.2.2 FDA:n COTS-vaatimusten lyhyt esittely

FDA:n (1998) ohjeessa todetaan, että ohjeessa käsitellään avainasioita, joita FDA:n tarkastajien tulisi laitevalmistajien anomuksista etsiä. Dokumentissa seuraavat kaksi käsitettä ovat oleellisia:

1. Minimaalinen vaara: Kun vioittuminen, väärä toiminta tai OTS (Off-The-Shelf) -ohjelmiston väärinkäyttö ei aiheuta minkäänlaista mahdollisuutta potilaan vakavalle vammautumiselle, OTS-ohjelmiston sanotaan aiheuttavan minimaalisen vaaran.
2. Merkittävä vaara: Kun vioittuminen, väärä toiminta tai OTS-ohjelmiston väärinkäyttö aiheuttaa todennäköisesti kuoleman tai vakavan vamman potilaalle, OTS-ohjelmiston sanotaan aiheuttavan merkittävän vaaran.

Kaikille lääkintälaitteen sisältämille COTS-komponenteille on täytettävä perusvaatimustiedot. Tarkan tunniste-, toiminta- ja resurssimäärittelyn lisäksi tässä vaiheessa vaaditaan dokumentteja, joilla luvan anoja osoittaa varmistuneensa COTSin oikeanlaisesta

toiminnasta, esim. testiraportit. Tämän jälkeen laitevalmistajan on suoritettava vaara-analyysi. Mikäli laite aiheuttaa pahimmillaankin vain minimaalisen vaaran, asettaa tämä ylärajan COTSin vaarallisuudelle ja tämä riittää tähän kohtaan. Muutoin joudutaan listaamaan laitteen vaarat, arvioimaan näiden vakavuus ja mahdolliset aiheuttajat. Mikäli tämän jälkeen voidaan todeta, että laitteen aiheuttama vaara on minimaalinen, voidaan prosessi lopettaa. Muutoin on jatkettava vaaran pienentämisvaiheeseen. Mikäli laitteeseen jää minimaalista suurempaa vaaraa, arvioidaan sitä suhteessa saavutettavaan hyötyyn.

FDA julkituo samantapaisen huolen valmisohjelmistojen käyttämisestä toimialueellaan kuin viranomaiset ydinvoima-alalla. Valmisohjelmisto, joka on tarkoitettu yleiseen käyttöön, ei ehkä ole sovelias lääkintälaitteiden ohjelmistoksi. Vastuu valmisohjelmistojen riittävän tehokkaasta suorituskyvystä ja turvallisuudesta on yhä edelleen lääkintälaitteiden toimittajalla, vaikka tällä ei olisikaan kontrollia valmisohjelmiston valmistajan käyttämiin kehitysprosesseihin.

FDA:n mallin mukainen kelpoistusprosessi (ks. kuva 9) pohjautuu ensisijaisesti vaara-analyyseista saatuun informaatioon. Malli ei ota edellä kuvatun ydinvoima-alan kelpoistusmallin tavoin kantaa kehitysprosessiin, vaan prosessinarviointi tulee esille laadittaessa perusdokumentaatiota. Vaara-analyysien tuloksena OTS-ohjelmisto sijoitetaan johonkin edellä esitettyyn kolmeen luokkaan siten, että riskin vähentämisen jälkeen hyväksyttävä OTS-ohjelmisto ei saa olla luokassa merkittävä vakavuustaso.

Ohje määrittelee OTS-ohjelmiston (kuva 9: kohta 1**)** yleisesti saatavaksi ohjelmistokomponentiksi, jota lääkintälaitteen valmistaja käyttää sovelluksessaan tuntematta täydellistä ohjelmiston kehityselinkaaren hallintaa. OTS-ohjelmistoille valmistaja laatii perusdokumentaation (kuva 9: kohta 2), josta saa käsityksen jokaisen OTS-ohjelmiston komponentin valmistajasta ja kehitysprosessista. Perusdokumentaatio laaditaan kaikille OTS-ohjelmistokomponenteille. Sitä laadittaessa vastataan mm. seuraaviin kysymyksiin (FDA 1999):

1. Mikä OTS-ohjelmistokomponentti on? Annetaan komponentin ja valmistajan tunnistetiedot sekä luetellaan mukana tullut dokumentaatio. Selostetaan OTS-ohjelmiston tarkoituksenmukaisuus lääkintälaitteelle ja tiedossa olevat suunnitterurajoitukset.
2. Mihin konfiguraatioon OTS-ohjelmisto validoidaan? Selostettava laitteiston spesifikaatio: prosessori, muistien koko, tietoliikenne, näyttö jne. Selostettava ohjelmiston spesifikaatio: käyttöjärjestelmä, ajurit, lisäyksiköt jne.
3. Kuinka varmistua siitä, että loppukäyttäjä on ryhtynyt asianmukaisiin toimenpiteisiin? Selostettava mm.

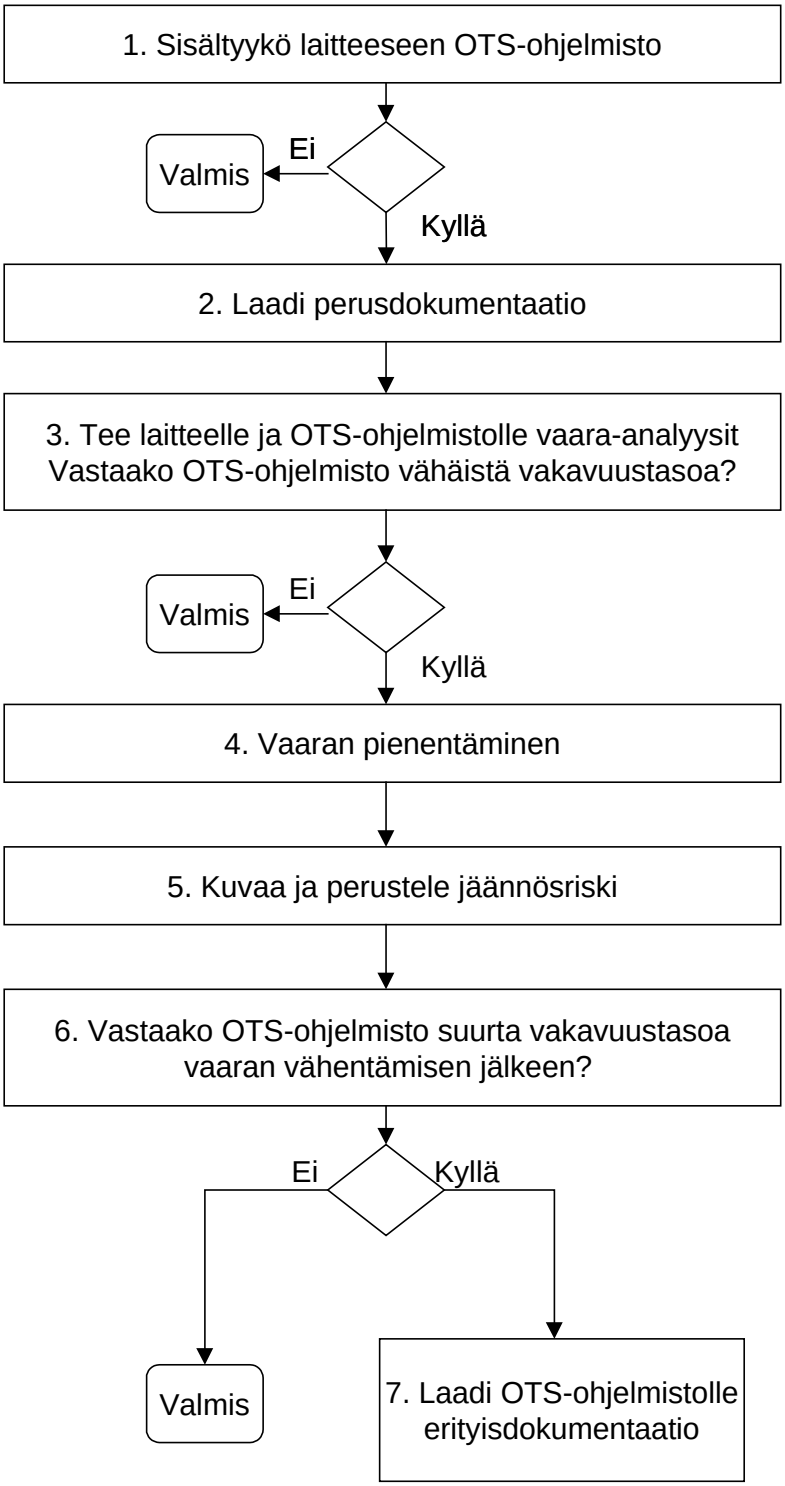
- mitä vaiheita tarvitaan tuotteen konfiguroimiseksi
- miten usein konfigurointia arvellaan tarvittavan
- mitä koulutusta käyttäjä tarvitsee
- mitä toimenpiteitä tarvitaan sen toteamiseksi, että lääkintälaite ei sisällä sinne kuulumattomia ohjelmistokomponentteja.

4. Mitä OTS-ohjelmisto tekee? Selostettava täsmällisesti ne ohjelmiston toiminnot, jotka sisältyvät lääkintälaitteeseen, mm. eriteltävät virhekäsittelyyn kuuluvat toiminnot. Lisäksi on selostettava mitä yhteyksiä ohjelmistolla on lääkintälaitteen ulkopuolisiin ohjelmistoihin.

5. Mistä tiedetään että OTS-ohjelmisto toimii oikealla tavalla? Kuvataan testit, verifioinnit ja validoinnit. Esitetään testitulokset ja selvitetään mahdollisesti ylläpidettäväksi tarkoitettu vikaluettelo.

6. Miten hallitaan OTS-ohjelmistotuotetta? Suunniteltava mm. seuraavat toimenpiteet:

   - toimenpiteet, joilla estetään väärien versioiden joutuminen lääkintälaitteeseen
   - OTS-ohjelmiston konfiguraation ylläpitotoimet
   - OTS-ohjelmiston säilytystapa ja paikka
   - asennustoimet
   - toimenpiteet, joilla varmistua OTS-ohjelmiston ylläpidosta ja koko elinkaaren kattavasta tuesta.



Kuva 9. OTS (Off-The-Shelf)-ohjelmiston päätöskaavio (FDA 1999).

Tunnusomaista lääkintälaitteiden kehitysprosesseille on jatkuva riskienhallinta, jossa vuorovaikutteisesti vaihtelevat vaara-analyysit ja niiden tuloksena vaadittavat riskien vähentämistoimenpiteet. OTS-ohjelmiston vaara-analyysit (kuva 9: kohta 3) ovat oleellinen osa lääkintälaitteen vaara-analyysia. Analyyseilla tunnistetaan merkittävät vioittumistavat, jotka voivat aiheuttaa vaaran potilaille, laitteenkäyttäjälle tai sivullisille. Lisäksi päätetään vaaran suuruudesta. Jos mikään tunnistetuista vaaroista ei ole suuruudeltaan merkittävä, tarkastelu lopetetaan ja dokumentoidaan analyysitulokset. Vähäisessä vakavuustasossa OTS-ohjelmiston dokumentaatioksi riittää siten perusdokumentti ja OTS-ohjelmiston vaara-analyysin tulosraportti.

Mikäli OTS-ohjelmiston vaara-analyysien tuloksena on merkittävään vakavuusluokkaan kuuluvia vaaroja, on suoritettava riskinvähennystoimenpiteitä (kuva 9: kohta 4) siten, että saavutetaan vähäinen tai merkityksetön vakavuusluokka. Riskiä voidaan vähentää sopivilla laitteisto- ja/tai ohjelmistotoimenpiteillä, jotka on jaettavissa kolmeen pääluokkaan:

- suunnittelu
- suojaustoimet
- varoitukset.

Näistä riskin vähennystoimista suositeltavinta on jo suunnittelussa eliminoida vaaratilanteen syntyminen. Suojaustoimilla ohje tarkoittaa passiivisia toimenpiteitä, joihin ei tarvita ulkopuolista apua. Vähemmän suositeltavaa ovat laitteenkäyttäjälle tai muille asianomaisille osoitetut varoittavat ohjeet.

Päädyttiin mihin vakavuusluokkaan tahansa, jäännösriskit on arvioitava (kuva 9: kohta 5) sekä seurattava jäännösriskien kehittymistä koko lääkintälaitteen ja sen OTS-ohjelmiston elinkaaren ajan. Jäännösriski on vaarakohtainen, kuten ovat riskin vähennystoimetkin. Kaikki OTS-ohjelmistosta tunnistetut vaarat luetteloidaan ja kunkin vaaran kohdalla toimeenpannaan riskinvähennys sekä selvitetään täydellisesti jäännösriskin ominaisuudet ja perustellaan hyväksyttävyys. OTS-ohjelmiston kohdalla jäännösriskiä verrataan vastaavaan totutulla tavalla kehitetyn ohjelmiston jäännösriskiin. Kaikki käyttökokemukset, jotka liittyvät sovelluksen kaltaiseen ympäristöön, esitetään ja katselmoidaan.

Jos jäännösriski on vielä riskin vähennystoimenpiteidenkin jälkeen korkea ja OTS-ohjelmisto kuuluu merkittävään vakavuustasoon (kuva 9: kohta 6), OTS-ohjelmistosta laaditaan erityisdokumentti (kuva 9: kohta 7). Erityisdokumentti sisältää seuraavat pääkohdat, joilla lääkintälaitteen valmistaja vakuuttaa FDA:n OTS-ohjelmiston riittävästä soveltuvuudesta:

- Vakuuttaa siitä, että OTS-ohjelmisto on kehitetty asianmukaisilla ja riittävillä työkaluilla, jotka soveltuvat OTS-ohjelmistoon tarkoitettuun lääkintälaitekäyttöön.
- Esittää, että toimintaohjeet ja verifiointi- ja validointitulokset ovat asianmukaiset ja riittävät lääkintälaitteen sekä turvallisuus- että suorituskykyvaatimusten osalta.
- Esittää, että asianmukaisin toimenpitein varmistutaan OTS-ohjelmiston vaatimasta jatkuvasta ylläpidosta sekä tuen tarpeesta.

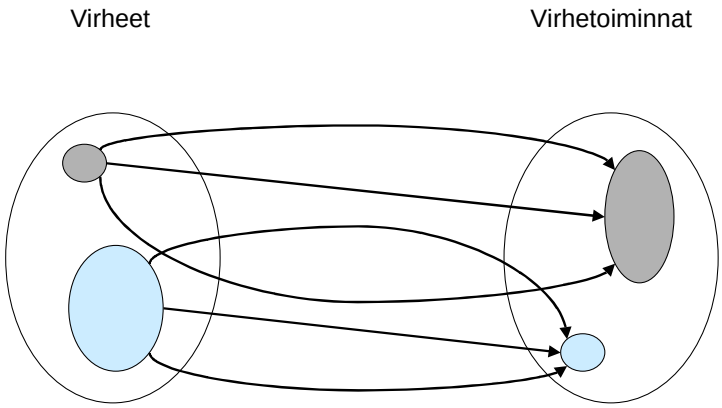
Käytännössä vakuuttaminen tapahtuu auditoimalla toimittajan kehitysprosessit. Verifiointi ja validointi sisältää OTS-ohjelmiston kehittäjän menetelmien lisäksi lääkintälaitteen valmistajan menetelmät, joilla kelpoistetaan OTS-ohjelmiston käyttökelpoisuus tietyssä lääkintälaitteessa. OTS-ohjelmiston ylläpitoa ja tukea tarvitaan erityisesti, kun alkuperäinen ohjelmistotoimittaja on lopettanut tukemisen.

5.3 Luotettavuusnäkökulmat

Edellisissä tämän luvun kohdissa käsiteltiin yleisellä tasolla valmisohjelmistoja ja niistä koostuvia järjestelmiä, mm. riskinvähentämistä ja yleistason arviointimenettelyitä. Niissä vain sivuttiin luotettavuutta. Valmisohjelmistoja ja ohjelmiston uudelleenkäyttöä on tarkasteltu hyvin monesta näkökulmasta ja kirjallisuusviitteitä onkin runsaasti. Luotettavuudessa on kyse sekä komponentin luotettavuudesta että niistä koostettujen sovellusten luotettavuudesta. Edelleen on kyse sekä luotettavuuden suunnittelusta että mittaamisesta. Luotettaviksikin todetut komponentit eivät välttämättä yhteenintegroituna merkitse luotettavaa sovellusta tai epäluotettaviksi todetut epäluotettavaa sovellusta.

5.3.1 Viat keskittyvät tiettyihin komponentteihin

Viimeaikaiset tutkimukset osoittavat useimpien ohjelmistosovellusten vikojen olevan peräisin vain muutamasta järjestelmäkomponentista (mm. Moller & Paulish 1993, Ohlsson & Alberg 1996). Näiden komponenttien tunnistaminen varhain edesauttaa projektia riskikontrollien asettamisessa. Kattavia analyyseja ja testejä voidaan kohdentaa korkean riskin omaaville komponenteille tai suunnitella komponentit uudelleen siten, että tietyt vioittumistavat eivät tule hallitseviksi. Vastaavanlaisia havaintoja on tehty aikaisemminkin tavallisille sulautetuille ohjelmistoille (kuva 10).



Kuva 10. Ohjelmistovirheet kasautuvat. Pieni joukko virheitä aiheuttaa näkyviä virhetoimintoja. Suurin osa virheistä johtaa harvoin virhetoimintaan.

Metriikkoihin perustuvia ennustemalleja on myös kehitetty. Emam et al. (2001b) lähestyvät ongelmaa kehittämällään olio-ohjelmoiduille järjestelmille kohdistetun metriikkapohjaisen ennustemallin. Koska se perustuu metriikoihin, heidän mukaansa sillä on myös pystyttävä validoimaan metriikoita. Tutkimus perustuu myös käytännön kokeisiin, joiden pohjalta tekijät väittävän mallin avulla kyettävän tarkasti tunnistamaan vialliset luokat ja arvioimaan ohjelmiston laatutasoa tulevien projektien tuotteille.

Heidän mallissaan metriikat perustuvat suunnittelun historiatietoihin ja ohjelmistokomponentit kuuluvat kaupallisiin Java-sovelluksiin. Nämä metriikat ovat joukko Chidamber & Kemererin (1994) ja Briandin et al. (1997) metriikoita, joista yhdistämällä saadaan yhteensä 24 erilaista metriikkaa. Niistä tutkijat valitsivat käyttöönsä kymmenen.

Kaksi Chidamber & Kemererin (1994) metriikkaa DIT (Depth of Inheritence Tree) ja NOC (Number Of Children) sisältyvät myös niihin metriikoihin, joita Briand et al. (2000) ovat arvioineet. Mahdollisen jatkotutkimuksen kannalta on kiinnostavaa arvioida miten oikeaan Emam et al. (2001b) ovat osuneet valitessaan kyseiset metriikat tutkimuksensa kulmakiveksi. DIT on pisin polku luokasta juureen periytymishierarkiassa. Mitä pitempi polku, sen monimutkaisempi luokka ja siten myös virhealttiimpi. NOC laskee niiden luokkien lukumäärän, jotka periytyvät tietystä luokasta. Briandin et al. (1997) kytkentämetriikat laskevat luokkien vuorovaikutusten määrän. Siinä määritellään käsitteitä kuten ystävyys ja periytyvyys.

Metriikoiden kehittäminen oliopohjaisille järjestelmille on osoittautunut ongelmalliseksi. Siitä kertoo jo runsas määrä viitteitäkin, mm. Lorenz & Kidd (1994), johon on koottu

n. 40 olio-ohjelmoinnin metriikkaa. Myös metriikkoja validoivia empiirisiä tutkimuksia on tehty runsaasti (mm. Basili et al. 1996, Briand et al. 2000, Fioravanti & Nesi 2000).

Toisessa julkaisussaan Emam et al. (2001a) käyttävät lähdekielisen ohjelman metriikoita ennustaessaan ohjelmistokomponenttien riskitasoa. Riskitason he määrittelevät korkeaksi silloin kun komponenteista on hyväksyntätesteissä paljastunut vika tai kun ne tarvitsevat erityisen paljon ylläpitoresursseja. Tutkimus perustuu laajaan reaaliaikaisen käytönaikaisen luotettavuustiedon määrään. Keskeisenä arviointikohteena ennustusmallissa on nk. CBR-luokittelija (Case-Based Reasoning), mikä käyttää vastaavia tapauksia komponenttien riskialttiiden luokkien ennustamisessa. Vastaavuus määritellään tuotemetriikoiden avulla siten että tietyssä luokkakannassa olevilla komponenteilla on samanlainen rakenne. Tutkimuksen lähtökohtana on ajatus, että jos ohjelmistokehittäjillä on edessään tällainen vastaavan luokan työ, myös mentaalinen rasite olisi vastaava ja siten virhealttius olisi samaa tasoa. Tutkimus keskittyy CBR-luokittelijan suorituskyvyn arviointiin. Omana arvionaan tutkijat toteavat, että luokittelijat eivät eroa suorituskyvyssä toisistaan niin selvästi, että monimutkaisempia luokittelijoita kannattaisi käyttää.

5.3.2 Luotettavuussuunnittelu

Komponenttipohjaisen ohjelmiston luotettavuusteoriasta ei ole julkaistu riittävästi tutkimustuloksia. Tutkimukset eivät keskity niinkään luotettavuuteen kuin ohjelmistokomponenttien hankintaan ja arviointiin osana integroitua järjestelmää. Integroitavuuden suunnittelu onkin mennyt selvästi ja ehkä aiheestakin edelle luotettavuussuunnittelua ja -arviointia tutkimustoiminnassa. Yleisin tapa kaikkien ohjelmistotyyppien luotettavuuden arvioimisessa on pohjata arviointi testauksista käyttöprofiilipohjaisesti saatuun luotettavuustietoon (ks. kohta 6.3.2).

Tarkastellaan tässä kohdassa valmisohjelmiston luotettavuussuunnittelun periaatteita, joista keskeisin on vertailu komponentin tilastollisen luotettavuustiedon ja komponentin riippumattomuusasteen välillä. Riippumattomuus on ohjelmistoille luotettavuusteknisten arviointien kannalta keskeinen ominaisuus, mutta se on erilainen (ks. alla) ohjelmistolle kuin laitteistolle. Yleensä ohjelmistossa ei ole montaakaan kohtaa, joka olisi täysin riippumaton – yhden muuttaminen muuttaa toista kohtaa saattaen aiheuttaa tahattomia sivuvaikutuksia.

Testaukset kohdistuvat sekä ohjelmistoon järjestelmänä että komponenttina. COTS-ohjelmistokomponenteista koostuvan järjestelmän luotettavuus tulisi arvioida myös komponenteista kootun luotettavuustiedon pohjalta. Tätä onkin lähestytty tutkimuksissa jo yli kahdenkymmenen vuoden ajan (mm. Littlewood 1979, Laprie 1984, Krishnamurthy & Mathur 1997). Menetelmä lähentäisi ohjelmistotekniikkaa perinteisiin tekniі-

koihin, joissa on totuttu laskemaan järjestelmän kokonaisluotettavuus sen komponenttien luotettavuustiedoista. Toisaalta, jos riippumattomuus voitaisiinkin osoittaa, tämä yksin saattaisi riittää osoitukseksi luotettavuudesta.

Markov-pohjaisilla malleilla on yleisesti laskettu komponenteista koostuvan järjestelmän luotettavuus. Mallit edellyttävät komponenteilta riippumattomuutta, mikä onnistuukin laitteistokomponenteilta, sillä ne suunnitellaan mahdollisimman riippumattomiksi. Jäljelle jäävät riippuvat tekijät otetaan mallinnuksessa huomioon. Valitettavasti laitteistolle sopiva riippumaton mallinnustekniikka soveltuu hyvin huonosti ohjelmistolle, sillä on lähes mahdotonta suunnitella riippumattomia ohjelmistokomponentteja. Jossakin määrin tätä on kuitenkin yritetty.

Woit & Mason (1999) esittävät konstruoimiaan suunnittelusääntöjä, joilla välttämätön riippumattomuus ohjelmistokomponenttien välille saataisiin aikaan. Riippumaton komponentti heijastaisi ohjelmiston tiedonsiirto-ominaisuuksia siten, että jokainen komponentti muodostaisi atomisen siirtymän syötteestä ulostuloon. Minkä tahansa komponentin luotettavuus olisi täysin riippumaton kaikista muista komponenteista, koska niiden välillä ei ole minkäänlaista suhdetta. Komponentit voivat herättää toisensa, mutta eivät hyödynnä toisen komponentin suoritustuloksia hyväkseen. Näillä periaatteilla järjestelmän luotettavuus on muodostettavissa (ks. Hamlet et al. 2001).

Yhdenmukaisuusperiaatetta voidaan hyödyntää, kun luotettavuustiedot vastaavasta samanlaisesta järjestelmästä ovat saatavilla sekä tiedetään miten luotettavuusarvot yhdistetään järjestelmän luotettavuuden laskemiseksi. Periaatteen soveltaminen riippuu ohjelmiston suunnittelijan kyvystä tunnistaa kaikki ne suunnittelutekijät, jotka heijastuvat aikaisemmista suunnitteluista. Myös luotettavuuden kasvumalleilla ennustetaan vanhojen ohjelmistoversioiden luotettavuustietojen pohjalta tulevien vastaavanlaisten ohjelmistoversioiden luotettavuutta. Luotettavuusmalleilla saataisiin arvioitua, saavuttaako ohjelmisto sille annetun luotettavuustavoitteen, tai mikä luotettavuus on odotettavissa testien päätyttyä. Jos nämä tiedot ovat käytettävissä aikaisessa vaiheessa, niillä kyettäisiin tukemaan resurssien allokointia ohjelmiston kehitysprojektissa. Yhdenmukaisuusperiaatetta tarkastellaan mm. lähteessä Mason & Woit (2000).

Hamlet et al. (2001) ehdottavat ohjelmistokomponenttiin tietynsisältöistä kylttiä , jossa ilmoitettaisiin komponenttia hyödyntävälle suunnittelulle välttämättömät tekniset luotettavuustiedot laskentoja varten. Kyltin laatimisen tulisi olla ohjelmistovalmistukselle kustannustehokasta. Se tulisi sisältää tiedot komponentin hyväksynnästä, ja tekijöiden mukaan myös satunnaistesteistä tarkasti määritellyllä käyttöprofiililla. Lisäksi kyltin tulisi sisältää ohjelmiston toiminnalliset tiedot.

Käyttöprofiili on eräs ohjelmiston luotettavuuslaskelmien edellyttämiä erityispiirteitä, mikä erottaa ohjelmistokomponentin elektroniikkakomponentin luotettavuusarvioinneista. Jälkimmäisissä komponenttien luotettavuus riippuu fysikaalisesta ympäristöstä. Lämpötila- ja jännitetiedot voidaan antaa riippumattomana käyttöehdoista, mutta ohjelmiston luotettavuudella on kriittinen riippuvuus jonkin muotoisista käyttöehdoista, jotka voivat ilmetä syötteiden käyttöprofiilina. Kehittäjä testaa aina komponentin jollakin käyttöprofiililla, mutta sulautettuna elektroniikkaan ohjelmistokomponentilla on toinen käyttöprofiili, mikä merkittävästi heikentää uskottavuutta kehityksenaikaisiin testauksiin.

Vaikka käyttöprofiiliin perustuva testaaminen on ohjelmiston luotettavuuden erityispiirre, se kuormittaa soveltamisosuutta, jota tulisi pyrkiä alentamaan. Jos ohjelmistokomponenttien luotettavuuskuvat olisivat käyttöprofiilista riippumattomia, niistä koostuvan järjestelmän integrointitestaukset helpottuisivat. Komponentin kehittäjällä olisi kaksi vaihtoehtoa riittävän luotettavuuskuvan saamisessa ilman käyttöprofiilipohjaisia testauksissa:

1) Täydellinen testaaminen, mikä edellyttää äärellistä syötejoukkoa.
2) Ohjelmien ajonaikaiset itsetarkistukset.

Yleensä ollaan sitä mieltä, ettei täydellinen testaaminen ole mahdollista, mutta etenkin kriittisissä sovelluksissa, missä syöteavaruus on äärellinen ja tunnettu, tämä on ja pitäisikin olla mahdollista. Knight et al. (1994) ovat jopa sitä mieltä, että täydellinen testaaminen on paljon luultua yleisempää. Ainakin joitakin ohjelman voidaan ajonaikaisesti tarkistaa satunnaisuuteen perustuvilla käyttöprofiilista riippumattomilla tekniikoilla (Blum & Kannan 1995, Ammann & Knight 1998).

Vigder & Dean (1997, 1998) ovat myös ovat koonneet joukon nyrkkisääntöjä COTSien luotettavaksi integroimiseksi. He olivat todenneet, että komponettiohjelmistojen kehittäminen on ollut voimakkaassa kasvussa luotettavuuden kustannuksella. Tuloksena on ollut järjestelmiä, jotka ovat virhealttiita ja vaikeita ylläpidettäviä. Heidän ratkaisunsa perustuu myös komponenttien riippumattomuuteen, mutta ei niinkään luotettavuuden mittaamiseen, vaan ongelmien eliminoimiseen huolellisella suunnittelulla ja konstruoinnilla. Säännöistä ensimmäinen on tärkein. COTSien kuorrutukseen perustuisivat monet muut heidän nyrkkisääntönsä. Sääntö perustuu siihen, että integroija pystyy valvomaan komponentin tietoliikennettä ja eristämään muut järjestelmän komponentit myös mahdollisilta kuorrutetun komponentin muutoksilta. Suunnittelu tulisi tehdä seuraavien periaatteiden mukaisesti:

1. Kuorruta kaikki komponentit.
2. Liitä komponentit sovellukseen riippumattomasti.

3. Todenna komponenttiversioiden yhteensopivuus.
4. Lisää assertioita ohjelmakuorrutukseen ja sovellusliitäntään.
5. Älä salli komponenttien vuorovaikutusta.
6. Yhteensovita avoimiin standardeihin.
7. Vältä liian aikaista sitoutumista arkkitehtuuriin.

Kuorrutuksessa *(kohta 1)* luotettava osa on ikään kuin “ohut kuori” epäluotettavan ohjelman ympärillä. Kuorrutus tarkkailee COTSin syötteitä ja ulostuloja. Mikäli nämä eivät ole sallitulla alueella, voidaan käyttäjää hälyttää ja/tai lopettaa ohjelman suoritus. Tällaisista menetelmistä voi olla hyötyä erityisesti silloin, kun järjestelmän jatkuva ohjauksen säilyminen ei ole kriittistä, vaan suoritus voidaan lopettaa välittömästi ongelmatilanteessa (lentokoneen ohjaus vs. röntgenkuvaus).

Kuorrutusten käyttökelpoisuutta rajoittaa se, että niitä on mielekästä tehdä vain melko yksinkertaisten ja paikallisten ongelmien tunnistamista varten. Myös kuorrutukset voivat tulla laajoiksi ja monimutkaisiksi ohjelmiksi, mikä saa kehittäjät luopumaan niiden käytöstä integrointia suunniteltaessa.

Kuorrutukset muistuttavat toiminnaltaan redundanssia, jossa perusajatuksena on, että ohjelmiston kriittinen osa voidaan jakaa luotettavaan, mutta yksinkertaiseen ytimeen ja toisaalta epäluotettavaan, mutta tehokkaaseen ja monipuolisempaan ulkokerrokseen. Ulkokerros ohjaa laitteen toimintaa ytimen tarkkaillessa systeemin tilaa jatkuvasti. Mikäli järjestelmän tila alkaa ajautua pois siitä tilajoukosta, jonka ydin kykenee hallitsemaan, ydin siirtää ohjauksen itselleen ja resetoi ulomman kuoren. Kun systeemi on saatettu turvalliseen perustilaan, palautetaan laitteen ohjaus taas tehokkaammalle ulkokuorelle. COTS-tapauksessa tämä voisi tarkoittaa sitä, että ostaja toteuttaa ostettavasta toiminnallisuudesta yksinkertaistetun version itse ja asettaa COTS-ohjelman ulkokuoreksi.

Yksi menetelmän ongelma on, että jostain täytyy löytyä luotettava ydin, mikä ei aina ole helppo tehtävä. Toiseksi käyttökelpoisuutta rajoittaa se, että järjestelmän tilan mittaamisen on oltava helppoa, nopeaa ja luotettavaa sekä myös se, että sallittu tilajoukko pitää pystyä selvästi rajaamaan. Jaon ytimen ja ulkokuoren välillä on istuttava sovellukseen. Ytimen ja ulkokuoren versionhallinta saattaa myös aiheuttaa omia haasteitaan järjestelmän kehittyessä.

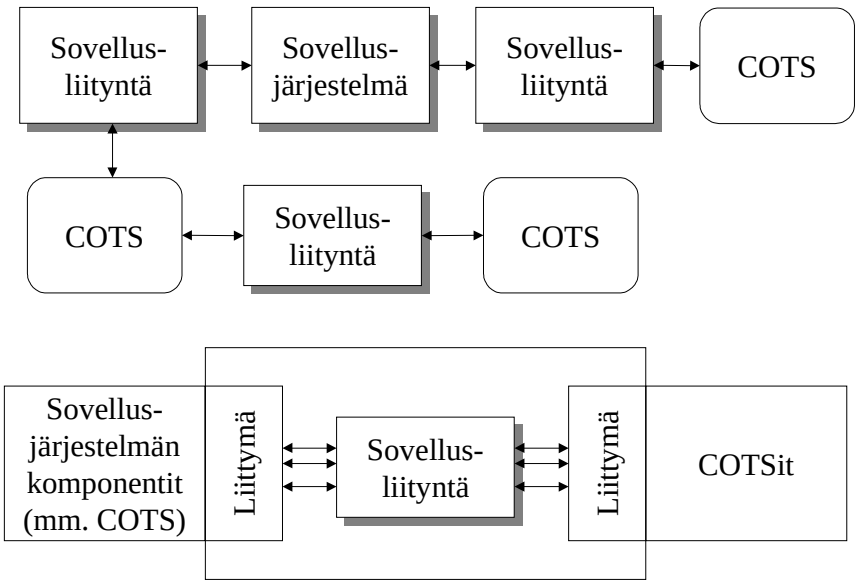
Redundattinen menetelmä muistuttaa puolestaan erilaisuusperiaatetta, mikä on yksi tapa vähentää ohjelmistosuunnittelun ja ohjelmoinnin aiheuttamien virhetilanteiden lukumäärää. Erilaisuusperiaatetta noudattaa mm. N-versio-ohjelmointi, jolla tarkoitetaan sitä, että ohjelman kriittisiksi todetuista lohkoista tehdään N (N = 2, 3 tyypillisesti) ver-

siota mahdollisimman riippumattomasti. Lisäksi ohjelmassa on oltava tarvittavat toiminnot eri lohkojen ajamiseen ja niitten tulosten vertaamiseen. Mikäli kaikki lohkot antavat saman tuloksen, voidaan jatkaa suoraan. Muutoin tilanteen mukaan suoritus voidaan lopettaa tai käytetään äänestysprosessia todennäköisimmän tuloksen löytämiseksi. COTSien tapauksessa tätä voitaisiin soveltaa hankkimalla eri valmistajilta COTS-ohjelmat samaan tarkoitukseen. Riippumattomuudelle asetetut vaatimukset toteutunevat, jos tuottajilla ei ole yhteyksiä. Erilaisuusperiaatteella komponenteista koottu sovellus on todennäköisesti luotettavampi kuin vastaava yksittäisellä luotettavalla ohjelmalla toimiva sovellus, sillä kyky havaita, käsitellä ja eliminoida virhetilanteita kasvaa. Toisaalta korkeissa luotettavuustavoitteissa tämäkään COTS-ratkaisu ei ole suotava.

Menetelmän haittoja ovat “ylimääräisten” COTSien lisenssimaksut, äänestysprosessin toteuttamisesta aiheutuvat kustannukset sekä useiden lohkojen ajamisen hidastava vaikutus. Usean COTSin käyttämisestä samassa ohjelmassa aiheutuvia riskejä on jo käsitelty edellä. Lisäksi voidaan aina kysyä, että toimiiko äänestysprosessi luotettavasti. On myös esitetty, että tietyntyyppiset virheet esiintyvät toisistaan riippuen, jolloin menetelmä ei takaa kaikkien virhetyyppien suhteen luotettavaa toimintaa.

Sovitusliittämisessä *(kohta 2)* keskeisiä toimintoja ovat tieto- ja ohjausvuot, keskeytysten hallinta ja tietomuunnokset. Sovitusohjelmia tarvitaan sekä COTSien yhdistämisessä keskenään että kototekoisten ohjelmistojen kanssa (ks. kuva 11). Tiedonsiirrollisten ominaisuuksien lisäksi sovellusliitäntäohjelmalla voidaan lisätä ja rajata COTSin toiminnallisuutta.

Komponenttiversioiden päivittäminen merkitsee myös ohjelmakuorrutusten, sovitusliitäntöjen ja muiden komponenttien päivittämistä sekä uudelleentodentamista *(kohta 3)*. Monet vaiheet on automatisoitu, mm. version hallinta ja komponenttien mahdollisten liittymien generointi, mutta vaadittavasta luotettavuustasosta riippuu, miten täsmällisiä yhteensovittamistestauksia tulisi tehdä. Assertioiden ja poikkeamatarkasteluiden lisääminen kuorrutuksiin ja sovitusliitäntöihin *(kohta 4)* lisää järjestelmän luotettavuutta nopeilla virhetilanteiden paljastuksilla ja eristämisillä. Assertioita voidaan asettaa koodiin hyvin yksinkertaisiin tehtäviin, kuten todentamaan parametrityyppejä tai -arvoja, tai monimutkaisiin tehtäviin, kuten tarkistamaan tapahtumien hetkellistä vaatimuksenmukaista järjestystä.



Kuva 11. COTS-pohjaisia järjestelmäkaavioita.

Kuorrutuksia ja sovitusliitäntöjä käytetään nimenomaan komponenttien riippumattomuuden edistämiseksi *(kohta 5)*. Tavoitteena on vuorovaikutusten eliminoiminen kokonaan tai oleellinen vähentäminen erityisesti luotettavuutta vaativissa sovelluksissa. Eliminoimisen hyödyt näkyvät myöhemmin muutostöiden yhteydessä, sillä vuorovaikuttavan komponentin suhde ympäristöönsä saattaa luotettavuudelle ratkaisevalla tavalla muuttua. Koska kehittäjä pääsee useimmiten käsiksi vain kuorrutuksiin ja sovitusliitäntöihin, kaikki ylimääräiset testi-, tarkistus-, eristämis- tai instrumentointivaatimukset lisätään vain niihin, ei varsinaisiin COTS-ohjelmistokomponentteihin.

COTS-komponentit, jotka on valmistettu yksinoikeudella kehitettyjen standardien mukaan, voivat aiheuttaa siirrettävyysongelmia ja mukautuvuusongelmia muiden valmistajien komponenttien kanssa *(kohta 6)*. Liian aikainen sitoutuminen arkkitehtuuriin *(kohta 7)* rajaa komponenttivalikoimaa ja lisää integraatio-ohjelmien (kuorrutusten ja sovitusliitäntöjen) ylimääräistä käyttöä.

5.3.3 Tietoturvasuunnittelu

Kaikilla COTS-tyypeillä, jotka on integroitu sovellukseen, voi olla tietoturvauhkia. Siksi komponenttien valmistuksessa tietoturvavaikutukset tulisi ottaa huomioon. Komponentti voi mm. sallia luvattoman käyttäjäsovelluksen tai resurssien ja palveluiden väärinkäytön. COTS-komponentissa voi lisäksi olla tahattomia tai tahallisia toimintoja (mm. Troijan hevosia). Jos komponentille sallitaan pääsy kriittisiin resursseihin tai se

suorittaa hyvin tärkeitä tehtäviä, soveltajan tulisikin tutkia huolellisesti ne osat komponentin rakennetta, joille oikeuksia myönnetään.

Kuten edellä jo todettiin, soveltajalla ei useinkaan ole pääsyä valmisohjelmiston suunnitteluun tai lähdekoodiin, mitä erityisesti tietoturvan kohdalla voidaan pitää myös positiivisena ominaisuutena. On mahdollista, että etenkin laajalevikkisen tuotteen kohdalla joku keksii ja kiertää tietoturvasuojat sekä levittää tuotetta markkinoilla. Toisaalta tuotedokumentoinnin informaation vajavuus voi haitata tietoturvan integrointia sovellukseen. Esimerkiksi kuorrutusohjelmia suunnitteleva voi jäädä tietämättömäksi joistakin komponentin tietoturvaan liittyvistä ominaisuuksista.

Tietoturvaa suunnitellaan riskiä välttävillä tekniikoilla. Lähestymistapana on ollut fyysinen erottaminen, tietovuoanalyysit sekä enemmän tai vähemmän formaalit verifiointimenetelmät. Virhevälttöisyys ei ole riittänyt ohjelmiston luotettavuustekniikassa enää pitkiin aikoihin, mihin Lindqvist & Jonsson (1998) ovat löytäneet kolme syytä:

1. Aikaisemmin kehitettiin pääasiassa asiakkaalle spesifisiä ratkaisuja. Ne tulevat kalliimmiksi ja kuluttavat enemmän tuotantoaikaa kuin kaupallisten valmisohjelmistojen hyödyntäminen.
2. Kryptografia tietoturvan tekniikkasovelluksena on osoittautunut vaikeasti toteutettavaksi.
3. Fyysisen erottamisen sijasta useimmat yritykset toivovat integroituvuutta ja sopivuutta internetiin.

On myös perusteltua, että koska monimutkaisissa järjestelmissä ei virheiltä voida täysin välttyä, rinnalle kehitetään virhesietoisia sekä tuotteeseen että suunnitteluun kohdistuvia tekniikoita. Suunnittelun on myös siedettävä mahdolliset kehitysympäristön muutokset, mikä on erityisen tärkeää juuri valmisohjelmistojen hyödyntämisessä, koska hyödyntäjillä ei ole hyviä mahdollisuuksia vaikuttaa kehittämiseen.

Kaikki komponenttiohjelmistot vaarantavat sovellusarkkitehtuurin tietoturvan. Siksi kaikki COTS:t ja niitä hyödyntävät arkkitehtuurit ovat tietoturvaan liittyviä komponentteja tai järjestelmiä. Kyse on myös tietoturvatasosta, jossa omistaja toimii. Tulisi määritellä tietoturvatasoja, joihin arkkitehtuurit ja siten myös COTSit luokitellaan. Jokaiselle tietoturvatasolle määriteltäisiin tietyt kokonaisarkkitehtuurin tietoturvavirheelle sietoiset menetelmät ja tietyn tietoturvan verifiointiprosessin läpikäyneet COTSit. COTSeilta tulisi siis löytyä valmistajan antamia osoituksia riittävästä tietoturvasta ja ohjeita osoitusten uudelleenverifiointiin.

Tietoturvariskit ovat peräisin useasta lähteestä. Niitä ovat ohjelmistokomponentin suunnittelu, hankinta, integrointi, internetyhteydet, käyttö ja ylläpito (Lindqvist & Jonsson

1998). Suunnittelun aikaiset tietoturvaongelmat ovat sovelluskehittäjälle tietämättömissä. Ongelmat voivat johtua bugeista, salaovista, viruksista, Troijan hevosista tai ylimääräisistä toiminnoista, joita kehittäjä ei ole osannut testauksissa ja tarkistuksissa ottaa huomioon.

Hankittaessa komponentteja kehittäjä voi samaistaa omat tietoturvavaatimuksensa valmistajan spesifioimiin vaatimuksiin päätyen mahdollisesti liian alhaiselle tasolle. Hän on voinut hankkia COTSin internetverkon välityksellä, jolloin joku kolmas osapuoli on saattanut päästä väliin aiheuttaen tietoturvaongelman. Integroitaessa ongelmia voi tulla tuotteiden erisuuruisista tietoturvatasoista tai riittämättömästä integrointivaatimusten tietämyksestä.

5.3.4 Ylläpidettävyyden suunnittelu

Ylläpidon tehtävänä on säilyttää koko COTS-järjestelmien palvelutaso sekä mahdollisesti toteuttaa päivitysluonteisia parannuksia. Omia ohjelmia voidaan jatkuvasti kehittää, mikä tyypillisesti tapahtuu siten, että kerätään tietoja vikaantumisista, etsitään ohjelmakoodista virheelliset kohdat, suunnitellaan tarvittava korjaukset, dokumentoidaan ja toteutetaan sekä asennetaan korjausversio asiakkaan järjestelmään. Kaikki tämä vaatii selvää käsitystä oman ohjelman toiminnasta, ohjelmiston on oltava organisaation tarkassa hallinnassa. COTSien tapauksessa näin ei useinkaan ole.

Vaikka ilmiselviä vikoja havaittaisiin, niitä ei yleensä pystytä korjaamaan. Lähdekielistä koodia ei mahdollisesti ole käytettävissä niin, että korjauksissa joudutaan tyytymään kiertoteiden etsintään ja vikatilanteiden estämiseen työskentelemällä COTSia ympäröivällä omatekoisella ohjelmakuorrutuksella. Tämä todennäköisesti heikentää laitteen palvelutasoa. Vaikka COTSin valmistaja myöntyisi tarvittaviin muutoksiin, kestävät nämä todennäköisesti kauemmin kuin itse tehtäessä. Siten COTS-pohjaisten järjestelmien ylläpidettävyyteen on kiinnitettävä ajoissa huomiota ja rakennettava sitä jo soveltamisprojektin alusta alkaen.

Järjestelmä useimmiten koostuu itse tehdyistä ohjelmakomponenteista ja kaupallisista valmiskomponenteista, mikä aiheuttaa ylläpidettävyysongelmia. Käyttötarve muuttuu ja esitetään voimistuvia toivomuksia tietyn toiminnallisuuden lisäämiseksi. Mikäli toimintojen toteuttaminen vaatii muutoksia COTSiin, on tilanne entistä hankalampi. Siitä selvitään hankkimalla uusi COTS-versio tai kilpailijan uusi tuote. Kummassakin tapauksessa joudutaan kuitenkin testaamaan ja varmistumaan koko COTSin luotettavuudesta uudelleen.

On myös tarkoin harkittava COTSin päivittämiseen liittyviä kysymyksiä. COTSin valmistaja saattaa väittää uuden ja vanhan version olevan täysin yhteensopivia edellisten toimintojen osalta. Kokemus on kuitenkin osoittanut, että näin ei aina käytännössä ole.

Hyvä ylläpidettävyys rakentuu toimenpiteistä, jotka kohdistetaan erilaisiin prosesseihin: komponenttien päivittäminen ja lisääminen, toimintojen täsmentäminen ja lisääminen, vianetsintä, järjestelmävalvonta sekä tuotteen hallinta.

Komponenttien vaihtaminen ja lisääminen edellyttää uudelta komponentilta soveltuvuuden arvioimista, sovellusliitännän ja ohjelmakuorrutuksen uudelleen koodaamista, testaamista ja tuotehallinnan erilaisia toimenpiteitä, joista versiohallinta on yksi merkittävimmistä. Nykyisissä järjestelmissä loppukäyttäjän päämielenkiinto on järjestelmien tuottamissa palveluissa. Palvelut tuotetaan sovellusjärjestelmässä, joka saattaa koostua useista erilaisista COTS-komponenteista ja sovelluksista. Kun palveluita lisätään tai muutetaan, vaikutukset yltävät komponenttitasolle ja ovat varmasti hankalia, jos niihin ei ole osattu etukäteen varautua järjestämällä komponentteja uudella tavalla.

Erityisesti järjestelmät, joissa on useita eri valmistajan sovelluksia, ovat alttiita ongelmille. On nopeasti selvitettävä ongelman syy, joka voi olla yksittäisessä COTS-komponentissa tai seurausta usean komponentin vuorovaikutuksesta. Ongelman selvittämistä vaikeuttavat lukuisat komponentit, eri valmistajat ja komponenttien mustalaatikkoisuus. Vianetsintä voi vaatia yhteydenottoja useaan komponenttivalmistajaan, jotka myös haluavat kuulla miksi juuri heidän ohjelmistokomponenttiansa epäillään ongelmanaiheuttajaksi. Vianetsintä helpottuu, jos järjestelmän valvonta on hoidettu riittävin toimenpitein, joista selviää, mitkä komponentit ovat käytössä, kuinka niitä on käytetty, missä vuorovaikutuksessa ne ovat sekä mitä ongelmia on esiintynyt.

Loppukäyttäjän näkökulmasta tuotteenhallinnalla olisi kyettävä selvittämään, mitä ohjelmaversioita on missäkin paikassa esillä, mikä niiden historia on, mitkä versiot ovat yhteensopivia sekä miten eri versiot eroavat toisistaan. Tämä tuotteenhallinta kohdistuu myytävään objektikoodiin, lisäarvoa tuotteenhallinta saa, jos siihen sisältyvät kehitystyökalujen sekä integrointiohjelmien, kuten sovellusliitäntöjen ja ohjelmakuorrutusten versiohallinnat.

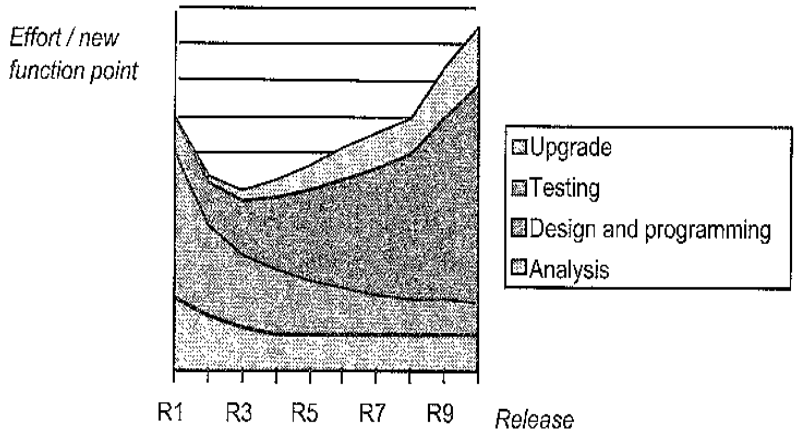
Hyvään ylläpidettävyyteen kuuluvat edellisten lisäksi myös monet muut tekijät, kuten konfiguroinnin yhdenmukaisuus, räätälöitävyyden helppous, komponenttien liittymien kontrollointi sekä avoimien standardien suosiminen (Vigder & Dean 1998). Niistä räätälöinnin helppous tulee usein esille asiakkaan vaatiessa alituisesti uusia järjestelmätoimintoja liiketoimintansa tueksi.

5.3.5 Monimutkaisuuden vähentäminen laajassa järjestelmässä

Inkrementaalinen prosessi on myös asiakkaitten toiveiden mukainen, sillä he eivät halua tuotetta jonka kehittäminen vie vuosia. Valmistajalle onkin tärkeää tehdä tuote, jossa on vain perustoiminnot, jos myyntimahdollisuudet ovat vielä avoimet. Jos kehitys kestää vuosia, voivat markkinat sillä aikaa hävitä.

Useimmat käytännön dokumentointiohjeet kattavat vain erikoiskehittämisen, mutta eivät tue inkrementaalista kehittämistä. Dokumentointiohjeiden puuttuminen lisää virhealttiutta. Inkrementaalinen ja iteratiivinen prosessi sopii toisaalta hyvin kriittisten järjestelmien kehittämiseen, sillä yleensä kriittisimmät ja merkittävimmät toiminnot toteutetaan ensimmäisinä.

Komponenttipohjainen suunnittelu ja yleensäkin valmisohjelmiston hyödyntäminen ovat siksi uutta, että niiden tulevaisuus nähdään ruusuisena. Kuvitellaan, että on helppo suunnitella ja ohjelmoida sekä lisätä toiminnallisuutta valmiista komponenteista. Totuus on kuitenkin toinen, kuten oheisesta kuvasta 12 nähdään.



Kuva 12. Ponnistukset jokaisen uuden toiminnon lisäämisen jälkeen (Jaaksi et al. 1999).

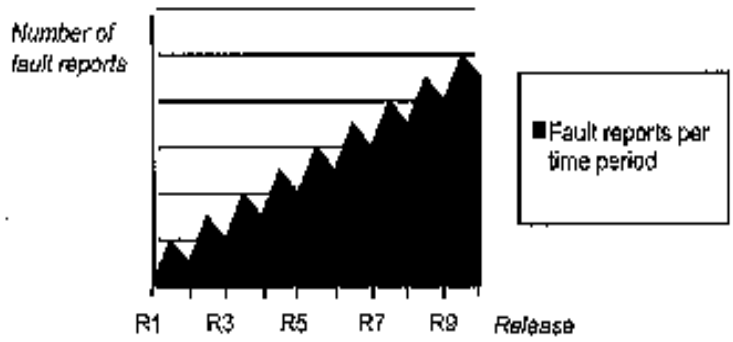
Tuottavuus uppoaa lisätesteihin, ylläpitoon ja päivitykseen. Vain analyysit pysyvät muutaman julkistamisvaiheen jälkeen vakiona. Jokainen julkistaminen merkitsee toiminnallisuuden kasvamista, mikä johtaa muutoksiin arkkitehtuurissa ja arkkitehtuuriosien riippuvuuksien lisääntymiseen, mikä kasvattaa monimutkaisuutta. Regressiotestit lisäävät testikustannuksia. Niissä testataan että toiminnallisuus on yhä voimissaan.

Järjestelmän ylläpidettävyyteen kuuluvat seuraavat asiat:

- suorituskelpoisen version vaihtaminen
- suorituskelpoisuuden lisääminen ja muuttaminen
- kokoonpanotiedostojen paikkaaminen
- tietokantakaavioiden päivittäminen
- back-upien ja muunnosten tekeminen.

Ylläpidettävyys voi uusissa julkistamisissa kasvaa voimakkaamminkin kuin mitä kuva esittää.

Toiminnallisuuden kasvattaminen on työlästä ja vaikeata. Sitä voidaan helpottaa parantamalla arkkitehtuurisuunnittelua siten, että arkkitehtuuri saadaan laajennettavaksi ja päivitettäväksi. Jaaksi et al. (1999) ehdottavat menetelmää, jossa komponenteilla on tiukka sisäinen ja väljä ulkoinen yhtenäisyys.



Kuva 13. Vikaraportit asiakkailta julkistuskohtaisesti. Tarkastelussa oletetaan, että ohjelmisto on tasalaatuinen, eli vikatiheys (vikaa / kloc) on vakio (Jaaksi et al. 1999).

Jokaisen julkistamisen jälkeen asiakkaiden ja käyttäjien määrä kasvaa. Kuvassa 13 pieni piikki merkitsee sitä, että kaikki asiakkaat ovat ottaneet tuotteen käyttöönsä. Palautteet tulevat heti käyttöön oton jälkeen ja laantuvat sitten. Vikojen määrä kasvaa joka julkistamisen jälkeen, koska sekä koko että käyttö kasvaa, ellei sitten ole jotain dramaattista kehittymistä tapahtunut tuotekehittelyn laadussa. Palautteissakin on kyse usein samasta ohjelmavirheestä.

Koon kasvaminen merkitsee uusia tapoja soveltaa systeemiä, ja siten riippuvuudet osien välillä kasvavat. Tämä merkitsee monimutkaisuuden ja virhelähteiden määrän kasvamista tehden testaamisen ja virheiden poistamisen entistä vaikeammaksi ja aikaavievämmäksi.

Laajoissa järjestelmissä tarvitaan eri tahojen tietämystä. Projektiorganisaatiot ovat isoja ja hierarkkisia. Kehitysprosessi on mutkikas, päätöstenteko tehotonta ja kokousten lukumäärä kasvaa. Monimutkaisuus kasvaa, koska riippuvuuksien hallinta on heppoista. Jos ei tunneta osien välisiä riippuvuuksia, ei ole mahdollista *ymmärtää muutosten aiheuttamia seurausvaikutuksia* järjestelmälle.

Mahdollisten riippuvuuksien lukumäärä kasvaa jyrkästi komponenttien kasvun myötä. Monet riippuvuudet ovat lisäksi luonteeltaan transitiivisia, mikä ennestään kasvattaa monimutkaisuutta.

Laajojen järjestelmien kehitysaika on pitkä, mikä johtuu luontaisesta isojen organisaatioiden tehottomuudesta ja järjestelmän monimutkaisuuden kasvusta. Pullonkaulat ovat projektin myöhemmissä vaiheissa, koska ne eivät ole riippumattomia toisistaan tai ohjelmiston kehittämistehtävistä:

- blackbox -testit laaditaan vasta, kun spesifikaatiot ovat käytettävissä
- glassbox -testit, kun suunnittelu on täydellinen
- asiakasdokumentointi, kun käyttöliittymät on jäädytetty
- asennus- ja päivitysohjeet, kun lopulliset tiedot ovat käytettävissä: tietokantakaaviot, kokoonpanotiedostojen käyttö ja rakenne jne.

Pienetkin muutokset kehitystyön alussa voivat aiheuttaa runsaasti lisätyötä elinkaaren myöhemmissä vaiheissa. Tarvitaan lisää kommunikointia, mikä taas on omiaan pitkittämään projektia. Resurssien lisäämisessä tyypillisiin pullonkauloihin hyödytään vain rajoitetusti.

6. Ohjelmiston luotettavuustakuut

Takuussa valmistaja vastaa siitä, että tuote toimii siten kuin on sovittu. Sopimus ilmenee käyttöohjeissa ja muussa dokumentaatiossa. Nykyinen käytäntö on antaa tuotteelle määräaikainen takuu, jona aikana korjataan esiin tulleet viat ja puutteet. On kuitenkin olemassa intressiryhmiä, joille ohjelmistopohjaisen järjestelmän luotettava ja turvallinen toiminta on erityisen tärkeää. Tässä luvussa tarkastellaan mahdollisuuksia nostaa takuurajoja siten, että ohjelmistonvalmistaja kykenee perustellusti lupaamaan asiakkaalle tai intressiosapuolelle korkeaa luotettavuutta ottamalla huomioon sanktiot ja juridiset näkökohdat. Lisätään valmiuksia tehdä ohjelmistotuotteita, joilla on paremmat menestymismahdollisuudet.

Ohjelmistokomponentteja valmistavalle ja käyttävälle teollisuudelle luotettavuustakuiden merkitys on selvä. Tämä koskee erityisesti yritystä, joka on aloittamassa toimintaansa ja saamassa ensimmäisiä tuotteitaan markkinoille. Myös yrityksen sisällä projektipäällikkö joutuu usein takaamaan ohjelmiston virheettömän toiminnan johtoryhmälle. Ohjelmiston julkistajana on usein yrityksen toimitusjohtaja, joka voi luottaa hyvin toimivaan ”takuuprosessiin” jo hyvissä ajoissa ennen tuotteen valmistumista. Asiakkaat eli tuotteen varsinaiset loppukäyttäjät ja omistajat ovat myös usein kiinnostuneita takuutodistuksista viranomaistahoja varten.

Seuraavassa luvussa tarkastellaan takuukäsitettä ohjelmiston kannalta. Luvussa 6.2 tarkastellaan takuita kolmannen osapuolen kannalta. Tavoitteena on selvittää mitä nämä lupaavat, jos heidän takuunsa pettävät. Asia on monimutkainen siinäkin mielessä, että sertifiointilaitos tai yleensä takuun antaja haluaa välttyä sanktioilta, mutta myös aikaa vieviltä sanktioprosesseilta. Viimeisessä kohdassa tarkastellaan testausten riittävyyttä takuiden pohjaksi.

6.1 Yleistä ohjelmistotakuista

6.1.1 Nykyiset takuusopimukset

Valmistaja vastaa vain siitä, mitä ohjelmasta on ilmoitettu. Jos ohjelma ei kykene suoriutumaan asiakkaan käyttötarpeista, ei takuuvastuuta synny, mikäli kyseistä toimintaa ei ohjelman dokumentaatiossa ole määritelty. Asiakas siis vastaa vahingoista, jotka syntyvät siitä, että hän on ostanut väärän tai liian vähän suorituskykyä omaavan ohjelman. Edelleen samasta syystä on tärkeää riittävällä tarkkuudella määritellä toiminnot, johon ohjelmaa voidaan käyttää.

Samoin on määriteltävä joko sopimuksessa tai ohjelman dokumentaatiossa ne laitteistot, joissa ohjelma toimii tehokkaasti, jolloin asiakkaan vastuulle jää puutteellinen toiminta

muissa laitteissa. Toimittaja ei vastaa myöskään siitä, että ohjelmat ovat täysin virheettömiä. Vastuu rajautuu edellä mainitulla tavalla siihen, mitä toimintoja tuotteen on luvattu suorittavan.

Takuuehdoissa voidaan määritellä missä ja miten takuukorjaus suoritetaan. Edelleen voidaan kirjata, miten takuun alaisesta virheestä on toimittajalle ilmoitettava ja mihin toimenpiteisiin asiakkaan on ryhdyttävä takuukorjauksen edesauttamiseksi.

Kyse on siis sopimusehdoista. CERD-projektin ”takuut-teemassa” tarkastellaan mitä (miten paljon) voidaan sopimuksissa luvata siitä, että ohjelmistolla on tietyn tasoinen luotettavuus, turvallisuus tai laatu. Tason ei siis välttämättä tarvitse olla korkea, kunhan lupaaja tietää miten taso saavutetaan ja mitataan. Lupaajia voivat olla esimerkiksi projektipäällikkö johtokunnalleen, valmistaja toimittajalle, toimittaja asiakkaalle ja asiakas kolmannelle taholle. Tähän teemaan eivät välittömästi kuulu ohjelmiston arvioinnin asiat, koska niitä käsitellään muissa CERD-projektin teemoissa, joita ovat COTSien luotettavuus, eheystasot ja luotettavuuden arviointi. Kuitenkin takuut-teema antaa viitteitä ja vaateita siitä, minkälaisia todisteita ohjelmiston luotettavuudesta halutaan. Jossakin määrin käsitellään tässä luvussa (6.3) testausten merkitystä takuusopimusten kannalta.

Jos luvataan paljon, tulee myös takuusopimusprosessissa ottaa huomioon monia asioita. Jatkotutkimuksessa tulisi tarkastella takuita kolmesta näkökulmasta:

1. sopimusehdot (miten todeta virheellisyys käytön aikana, vertaaminen valmistajan todisteisiin)
2. ohjelmistoseikat (ylläpidettävyys, muutokset, eheys)
3. juridiset seikat (lainsäädäntö ja käytäntö).

Sopimusehdoissa tulisi selvästi kirjata ylös, miten havaitaan, että sopimusehdot eivät täyty. Pitää ilmaista mistä virhetoiminnoista ja ohjelmistoympäristöstä on kyse. Monissa ohjelmistoissa on ollut vakavia virheitä, jotka ovat tuottaneet asiakkaille taloudellisia menetyksiä. Osittain tämä johtuu siitä, että testaamalla ei kyetä täysin osoittamaan ohjelmiston virheettömyyttä. Valmistajan tulisikin ennen kaikkea ilmaista sopimusehdoissa, että ohjelmiston toiminnat ovat tiettyä luotettavuustasoa samassa ympäristössä kuin missä se on testattu ja samoilla testitapauksilla. Vain näillä rajoitteilla ohjelma on tiettyä luotettavuutta käytön aikana. Kuitenkin hyvistäkin todisteista huolimatta ohjelmistoon voi jäädä vakavia virheitä ja silloin juridisen puolen tulisi olla kunnossa.

Toimittajan pitää taata ohjelmiston koskemattomuus siten, että joko ohjelmistoa ei voida toimittajan tai valmistajan tietämättä muuttaa tai jäljet muuttamisesta saadaan näky-

viin. Jos ohjelmisto vaatii päivittämistä tai korjausta, niin tulisi sopia miten ohjelmiston eheys säilyy.

Sopimusehdoissa viitataan yleisesti käyttöohjeisiin ja muuhun ohjelmiston dokumentaatioon. Ehkä yleisimmät asiakastuen valitukset kohdistuvat siihen, etteivät asiakkaat lue käyttöohjeita. Usein kuitenkaan asiakkaat eivät saa selvää käyttöohjeista, ja niissä tapauksissa käyttöohjeet saattavat olla puutteelliset. Hyvän dokumentaation kirjoittaminen maksaa. Erään tiedon mukaan (Kaner & Pels 1998) yhden sivun kirjoittaminen vie yhden työpäivän. Ehkä valmistajat eivät halua investoida käyttöohjeisiin, koska niin monet käyttäjät eivät kuitenkaan lue niitä, vaan kääntyvät asiakastuen puoleen tiedostamatta sitä, että monet asiakkaat yrittävät ratkaista ongelmat ilman asiakastukea.

Markkinointiaineistolla ei ole yleensä niin suurta merkitystä kuin esimerkiksi käyttöohjeilla sopimusehtoina. Niitä ei myöskään tarkisteta siinä määrin kuin muita ohjelmistodokumentteja. Joskus lupaukset ovat tahallisesti ylisuuria, joskus tahattomasti. Ei ole helppoa ratkaista, kummasta on kyse.

6.1.2 Juridiset näkökulmat

Lainopillisesti katsottuna erotetaan kolme erilähtöistä riskinottoa:

1. Kaupallinen riski
2. Oikeusopillinen riski
3. Tekninen riski

Kaupallinen riskinotto on pikemminkin riskin jakamista kuin riskin vähentämistä. Otetaan vakuutuksia ja sopimuksellisesti ja lakiin vedoten suojataan omaa osuutta virhetoimintojen tapahduttua.

Oikeusopillisessa riskinotossa keskeiselle sijalle nousevat rankaisuvaatimukset, jotka ovat sopimusten yläpuolisia asioita. Niihin kuuluvat huolimattomuus, välinpitämättömyys, tahallisuus ja petoksellisuus. Tahallisesti tehtyjen ohjelmistovirheiden määrä on erään tutkimuksen mukaan selvästi lisääntynyt yrityksissä (Laprie 1999).

Huolimattomuuden osoittaminen voi olla hyvin vaikeaa. Tuskin yhdellä tai kahdellakaan seikalla kyetään todistamaan valmistavan yrityksen toimineen huolellisesti tai huolimattomasti. Laatujärjestelmät ja niiden sisältämät hyvät toimintaohjeet ovat tässä perusedellytyksiä. Esimerkkejä on paljon, kuten mm. seuraavat kysymykset osoittavat:

- oliko ongelma tiedossa?

- olivatko turvallisuusanalyysit hyvin tehtyjä?
- kuinka hyvin suunniteltu on ohjelman virheen käsittely?
- miten asiakkaan palautteet on käsitelty?
- mikä oli testikattavuus?
- onko noudatettu suunnittelu- ja kehitysstandardeita?
- mitä dokumentaatio esittää riskeistä?

Useimmiten oikeutta ei kuitenkaan käydä, jos kyse on huolimattomuudesta tai välinpitämättömyydestä eikä sopimuksen rikkomisesta, petoksesta tai tahallisuudesta.

Kuten tietoturvan kohdalla on havaittu, lainsäädäntö on selvästi käytäntöä jäljessä. Osapuolet joutuvat ottamaan *teknisiä juridisia riskejä*, luotettavuuden ja turvallisuuden (kriittisten virhetoimintojen) kohdalla sitäkin enemmän. Lainsäädännön puutteellisuus merkitsee myös sitä, että asiaan perehtyneitä lakimiehiä ei ole riittävästi. Juridiikan mukaan tulo ohjelmiston laatuun lisää varmasti kustannuksia, mutta juridiikka tulee mukaan joka tapauksessa. Siksi takuumenetelmien tulisi olla hyvissä ajoin valmiina.

Lakeja ei voida säätää niin tarkasti, että ne kattaisivat kaikki tekniset yksityiskohdat ja tapaukset. Oikeuden päätökset vaihtelevat vastaavista tapauksista. Valmistajat ja ostajat varmasti tuntevat kokevansa vääryyttä, vaikka ohjelmiston laatuun ja luotettavuuteen kohdistuvia lakia olisikin riittävästi.

Juridiikkaan kuuluu ehdottomasti täsmälliset määrittelyt, jotka erityisesti ohjelmiston laadun ja luotettavuuden kohdalla ovat hajanaiset. Yksi vaikeimmin määriteltäviä on ohjelmistovirhe. Oletetaan, että määritellään virhe poikkeamiksi spesifikaatiosta. Mitä tapahtuu, jos sattuu jotakin sellaista, jota spesifikaatio ei kata? Laki ei voi määritellä sitä mitättömäksi tapaukseksi eli lain mukaiseksi. Toisaalta, jos määritellään virhe miksi tahansa ohjelman käytön epäonnistumiseksi, nousee sellaisia kysymyksiä kuin ”epäonnistuminen kenen käyttäjän kohdalla, mikä käyttö, koska?” Virheen määrittelyn laventaminen laventaa käsitteitä muualla. Ehkä lakiperusteinen ohjelmiston virheellisyyden määrittely riippuukin valmistajan ja ostajan välisestä suhteesta. Ne ostajat, jotka eivät ole olleet tekemisissä ohjelman kehitysvaiheiden kanssa tai edes koskaan nähneet ohjelman määrittelyasiakirjoja, ovat eri asemassa kuin ostajat, jotka ovat itse määritelleet tai vaatineet ohjelman toimintoja ja ominaisuuksia, mm. käyttöliittymää.

Jos asiakas määrittelee hyvin monimutkaisen käyttöliittymän ja myyjä toimittaa määrittelyä vastaavan ohjelman, onko myyjän syy, jos ohjelman käyttäminen on vaikeaa ja aiheuttaa siten ongelmia?

On kuitenkin tapauksia, joissa ohjelma on selvästi virheellinen. Ohjelma ei suoriudu esitteissä tai käyttöohjeissa kuvatuista toiminnoista ja muista asioista, ohjelma kaatuu suorituksen aikana, hävittää tai korruptoi asiakkaan dataa, laskee virheellisesti jne.

Ohjelma voi olla virheetön, mutta käyttökelvoton. Se tekee sen mitä on odotettukin raivostuttavan hankalalla ja aikaa vievällä tavalla. Ohjelmaa ei voida välttämättä pitää virheellisenä sen takia ettei siitä pidetä. Suunnittelusta ollaan yleensä montaa mieltä. Ohjelmaan on saatettu jättää sellaisia ominaisuuksia tai rajoitteita, jotka ohjelmasuorituksessa toimivatkin käyttäjää harhauttavina ansoina johtaen sekä virhetoimintoihin että aikaavieviin ylimääräisiin toimenpiteisiin. Dokumentaatio voi olla myös puutteellista. Siinä ei ehkä ole virheitä, mutta sen luettavuus voi olla heikkoa tai siinä ei ole riittävästi yksityiskohtaisia ohjeita.

Ohjelmiston juridiikkaa, sopimusehtoja ja takuita tarkasteltaessa tulisi aina ottaa huomioon se tosiasia, että ei ehkä koskaan voida valmistaa virheetöntä ohjelmaa. Virheitä voidaan vähentää, mutta vaikeuksia tulee aina olemaan jäljellä olevien virheiden esiintymistodennäköisyyttä määrättäessä. Takuun myöntämisessä on kyse juuri tästä todennäköisyydestä. Todennäköisyysarvioita asetettaessa tulisi tarkastelun kohdetta kyetä riittävästi kaventamaan. Jos lainsäädännössä mennään liian pitkälle ja vaaditaan ”ehdottoman” luotettavia ohjelmia, ohjelmistovalmistajat joutuvat varmasti koville ja yrityksiä kaatuu pienestäkin virheestä.

Virheiden merkityksetkin vaihtelevat sen mukaan, onko kyse massamarkkinoista vai vain räätälöidyistä ohjelmista. Laajoissa markkinoissa datan tilapäinen häviäminen ei aiheuta yksittäiselle asiakkaalle suuria tappioita (mm. ylimääräistä työaikaa), mutta kun lasketaan kaikki asiakkaat yhteensä, tappiot voivat olla mittavia. Juridisesti on tässä tapauksessa merkittävää, mitä varoituksia mahdollisista virhetoiminnoista on asiakkaalle annettu. Valmistajan kannalta kyse on riskinotosta: panostaako varoituksiin, jotka voivat vähentää tuotteen menekkiä, vai luottamukseen siitä, että mitään vakavaa ei tapahdu.

6.2 Kolmannen osapuolen sertifioinnit

Ohjelmiston luotettavuutta on vaikea saavuttaa ja vielä vaikeampaa arvioida. Ohjelmiston luotettavuuden selvittäminen on epäkäytännöllistä. Korkeaan ohjelmiston luotettavuuteen yltäminen vaatii työläitä testijärjestelyitä (mm. oraclet ja testitapausten määrittelyt) ja miljoonia testikertoja. Koska arviointi on epäkäytännöllistä, on kehitetty vaihtoehtoisia menetelmiä.

Ohjelmiston sertifiointi määritetään tässä yhteydessä seuraavasti: Sertifioinnissa analysoidaan ohjelmistoa, kunnes on saavutettu riittävä vakuuttautuminen siitä, että ohjel-

miston luotettavuustavoitteet on osoitettu saavutetuiksi tai on osoitettu ettei luotettavuustavoitteita ole saavutettu.

Teknisesti ohjelmiston luotettavuuden tai laadun arviointiongelmaa voidaan käsitellä kolmella tavalla:

1) henkilösertifiointi
2) kehitysprosessin organisaatiotason laatusertifiointi
3) ohjelmiston tuotteen tekninen arviointi.

6.2.1 Henkilösertifiointi

Henkilösertifiointi voidaan tehdä useammalla tavalla. Perusajatus on hyvä, koska ohjelmistokehittäjän taito ja kokemus ovat merkittävimpiä ohjelmiston luotettavuuteen vaikuttavia tekijöitä. Sertifiointikin riippuu henkilön tehtävien kriittisyydestä. Ammatillisia tutkintoja, käytännön kokemuksia ja saavutettuja meriittejä voidaan pitää yksinä perusteina.

Tunnetaan myös yrityksiä, joiden lisensiointiehdot ohjelmiston kehittäjälle ovat tiukat, esimerkiksi seuraavat:

- tutkinto tekniikan, tietotekniikan tai matematiikan alalta
- ainakin 16 vuoden kokemus alalta
- referenssit ainakin yhdeksältä ihmiseltä, joista viiden täytyy olla lisensioituja
- vaadittavat erityisalan referenssit.

Microsoft väitti erään tiedon mukaan (Ayala 1998) sertifioineensa satoja tuhansia ihmisiä joko asiantuntijoiksi, sovelluskehittäjiksi, kouluttajiksi tai systeemisuunnittelijoiksi (Microsoft Certified Systems Engineer, MCSE). Myös Euroopassa on sertifiointielimiä, jotka myöntävät MCSE-arvoja. Osa sertifiointitoimintaa harjoittavista organisaatioista hakee asiakkaidensa luottamusta akkreditoinnin avulla, mistä Euroopassa ja kansainvälisesti on tullut yhä tärkeämpi pätevyyden osoittamista edistävä tekijä. Akkreditoinnin avulla sertifiointielin voi osoittaa toimintansa ja antamiensa todistusten, sertifikaattien uskottavuuden ja luotettavuuden.

Sertifioidun ohjelmistokehittäjän rinnastaminen korkealuokkaiseen ohjelmistoon ei kuitenkaan vaikuta hyvältä ajatukselta jatkotutkimuksen kannalta. Henkilökoulutus ja -arviointi parantavat ohjelmiston laatua ja luotettavuutta, mutta takeeksi vakuuksille niistä ei ole.

6.2.2 Prosessin laatusertifiointi

80-luvun puolivälissä tuotteen laadun suorat arviointimenetelmät, mm. testaukset ja formaalit verifioinnit, eivät tuntuneet sopivilta. Heräsi ajatus prosessin arvioinnista, mikä sitten johtikin prosessilähtöisten laatustandardien ja -ohjeistojen, mm. SFS-EN ISO 9000-3, DO 178-B, FDA 510(k), CMM (Capability Maturity Model), Bootstrap ja Spice kehittämiseen. Perusajatus oli että hyvä prosessi tuottaa hyvän ohjelmiston.

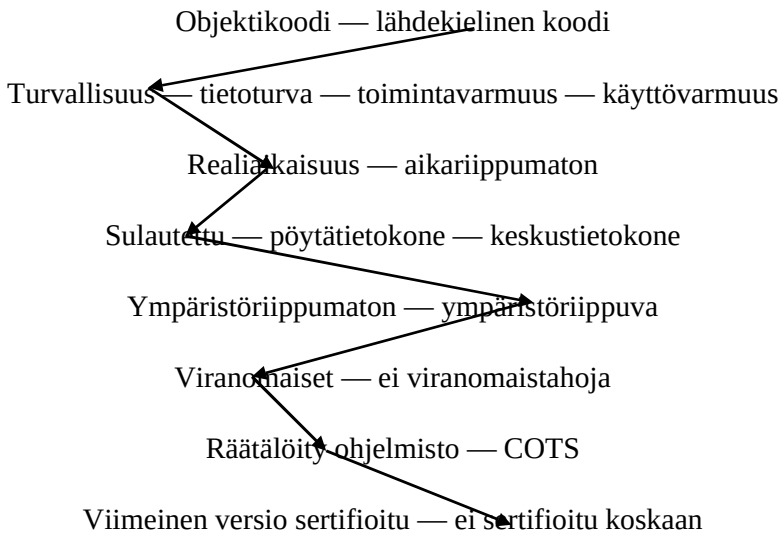
Takuiden kannalta prosessilähtöisissä standardeissa on se ongelma, että ne kohdistuvat laadun suunnitteluun ja saavuttamiseen, eivät arviointiin. Standardit vaativat mittaamista ja testaamista sekä keskittyvät kehitysryhmän sisäiseen infrastruktuuriin sen sijaan että tarkemmin ohjeistaisivat mitta- ja testivälineiden valintaa eri tasoilla.

Jotkut laatujärjestelmistä myös luokittavat ohjelmistokehityksen tietylle tasolle. Ongelmana on että hyvätkään kehitysprosessit eivät kuitenkaan voi taata hyvää ohjelmistoa. Hyvä prosessi lisää onnistumisen todennäköisyyttä ja huono vähentää. Eräät viranomaistahot eivät myöskään pidä prosessiarviointia suoran tuotearvioinnin korvaajana.

Ohjelmiston kehitysprosessin laatusertifioinnin rinnastaminen korkealaatuiseen tai luotettavaan ohjelmistoon ei voida pitää jatkotutkimuksen arvoisena, jos lähtökohdaksi otetaan ohjelmistotakuut. Ohjelmiston luotettavuutta laadun arviointi varmasti parantaa henkilösertifioinnin tapaan.

6.2.3 Tuotteen arviointi

Ohjelmiston kehitysprosessin arvioinnin heikkoutena on liian universaalinen lähestymistapa kaikkiin ohjelmiston kehitysmalleihin. Tuotearvioinnissa tästä päästään pakostakin eroon. Ohjelmistotuotteen tekninen arviointi on n-dimensionaalinen prosessi, jossa on otettava huomioon mm. kuvan 14 esittämät vaihtoehdot. Erilaisia vaihtoehtoja tuotteen arvioinnille on kuvan esimerkkitapauksessa $2 \cdot 4 \cdot 2 \cdot 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 768$ kappaletta.



Kuva 14. Ohjelmiston tuotearvioinnin lähtökohdat.

Ohjelmistotuotteen[8] laadun arvioinnissa on kaksi yleisintä ja ensisijaista lähestymistapaa: white box ja black box -testit. Black box -testeihin pitäisi sisällyttää myös luotettavuustestit. Musan (1998) standardimenetelmä ohjelmiston luotettavuustekniikaksi perustuu myös osin black box -testauksiin. White box -testeihin kuuluvat erilaiset arviointitekniikat, joilla joko perustellaan yksikkötestien kattavuutta tai haetaan staattista metriikkatietoa lähdekielisestä koodista.

Varsinaiseksi patenttiratkaisuiksi kummastakaan arviointitekniikasta ei ole. Esimerkiksi ei täysin ymmärretä, millainen suhde koodin monimutkaisuusmetriikalla on ohjelmiston luotettavuuteen. Luotettavuuden määritelmä viittaa loogiseen virheettömyyteen ja käyttöympäristöön, ei rakenteellisiin ominaisuuksiin. Myöskään ei mm. kyetä täydellisesti testaamaan yksinkertaista ohjelmaa, joka lukee 32 bitin kokonaislukuja.

Puuttuvien rakenteellisten tietojen (mm. lähdekoodi ja suunnitteludokumentaatio) käyttäjä ei voi hyödyntää white box -tekniikkaa COTS-ohjelmiston arvioinnissa. Valmistajat ja toimittajat pystyvät tähän, ja lisäksi he voisivat tarvittaessa käyttää kolmannen osapuolen sertifiointiapua.

Yksittäisenä toimenpiteenä näistä kolmesta ehdotetusta lähestymistavasta varmasti . tuotteen arviointi on sopivin perusta ohjelmiston laadun ja luotettavuuden sertifiointiin. Tuotteen dynaamisen toiminnan selvittäminen sopii erityisesti COTS-ohjelmistoille,

[8] Ohjelmistotuotteeksi tulkitaan myös vaihetuotteet, mm. määrittely- ja suunnitteludokumentit.

mutta ongelmia riittää. Tekniikoita tulisi olla useita ja laajalta alalta. Niillä pitäisi tarkastella sekä henkilöiden taitoja, prosessin kypsyyttä että tuotteen arviointia. Testausten ja tarkastusten kohdentamista kriittisiin ohjelmistokohtiin tulisi tarkastella. Tulisi laatia yhteinen ohjeisto, jossa otetaan huomioon kaikki kolme esitettyä lähestymistapaa.

Ohjeistetaan, miten painottaa arvioinnissa eri lähestymistapoja, esimerkiksi milloin ohjelmistokehittäjän taidon mittaaminen on merkittävä arviointilähde , milloin taas monimutkaisuuden mittaaminen ja miten nämä kaksi korreloivat toistensa kanssa. Voidaan olettaa, että jos ohjelmiston kehitysprosessi on tietyllä kypsyystasolla (mm. CMMI, Spice) ja kehityshenkilöt on sertifioitu tälle tasolle, tämä taso on pohja tuotteen sertifioimiseksi. Tuotteen sertifioiminen olisi silloin peruslähestymistapa takuutodistukselle. Menetelmällä on heikkoutensa, koska se voi vaatia subjektiivisten ominaisuuksien kvantifioimista (mm. henkilösertifiointi).

6.2.4 Sertifiointilaitosten lähestymistavat

Riippumattomille ohjelmiston laadun sertifioijille on markkinoita seuraavista syistä (Voas 1998b):

1. Ohjelmistovalmistajat eivät mielellään ota täyttä vastuuta omien ohjelmistojensa laadusta.
2. Kuluttajat haluavat tasapuolisia ja riippumattomia arviointeja.
3. Sertifiointikustannuksia voidaan vähentää ohjelmiston mittaamiseen erikoistuneiden yritysten välityksellä.

Viitteen mukaan ohjelmiston valmistajia pelottaa takuiden antamisessa erityisesti USA:ssa oikeuskäsittelyyn joutuminen ja valmisohjelmistojen (COTS) kohdalla erityisesti integroituminen kolmannen osapuolen ohjelmistoon. Sertifiointilaitoksesta on apua vakuuksien myöntämisessä, mutta ongelma siirtyy silloin sertifiointilaitoksen omalle kontolle, mikä ei tietenkään ollut alkuperäinen tarkoitus. Voas (1998b) antaa joitakin ohjeita joilla välttyä oikeudenkäynneiltä:

- Arvioitaessa testausten kattavuutta kannattaa määritellä, mitkä osat koodia on testattu, ei kannata ilmaista, että koodi, jota ei ole läpikäyty, on merkityksetön. Näistä sanoista jää helposti kiinni.
- Tulisi käyttää ilmaisua suhteellinen luotettavuus absoluuttisen luotettavuuden sijasta.

– Ei saisi hyväksyä periaatteella kaikki tai ei mitään, vaan esittää pätevät mittaustulokset. Asiakas saa päättää, riittävätkö todelliset sertifiointilaitoksen raportoimat mittaukset heidän käyttöönsä.

Voas (1998b) uskoo, että sertifiointilaitos kykenee mittaamaan ohjelmiston laatua sekä subjektiivisin että objektiivisin mittauksin. Kaikki mittaukset on kuitenkin tulkittava vain objektiivisesti. Ohjelmiston sertifiointilaitoksen etuna on tasapuolisuuden tuominen kilpailuun. Kaikki tuotteet saavat samanarvoisen käsittelyn edellytyksellä että on olemassa standardoituja mittareita.

Tarkastellaan olemassa olevia ohjelmiston sertifiointiorganisaatioita sekä sitä, mistä ne vastaavat. Tarkastelukohteita voisi valita useampia, mutta Underwriter's Laboratory (UL), NASA:n Independent Verification & Validation Facility ja National Security Computer Association kuvaavat hyvin tällä alalla toimivia yrityksiä. Tarkastelun tavoitteena on selvittää sertifiointilaitoksen toiminta yleisten ohjelmistotakuiden kannalta.

Ohjelmiston valmistaja voi hyödyntää kolmatta osapuolta, esim. ohjelmiston sertifiointi- tai testauslaboratoriota, tuotteensa luotettavuuden ja toiminnallisuuden osoittamisessa. VTT saa silloin tällöin kyselyitä, voisiko se antaa todistuksen tietyn ohjelman vaatimustenmukaisuuden osoittamiseksi. Vaateet valmistajilta, toimittajilta ja asiakkailta ovat lisääntymään päin. Toimittajat eivät mielellään vastaa myydyistä ohjelmistotuotteista, asiakkaat haluavat varmempaa tietoa kuin myynnin esittämät houkuttelevat lupaukset. Mitä seuraa, jos takuumies sanoo ohjelman olevan ”hyvä”, mutta se ei olekaan. Asiaa on tarkasteltu viitteessä Voas (1998b).

Voas luettelee joitakin sertifiointilaitoksia USA:ssa ja epäilee Underwriter's Laboratory'n jo lupautuneen sellaiseksi. UL toimii FDA:n eli lääketeollisuudessa ja lääkintälaitteiden alueella. UL on VTT Automaatiolle tullut tutuksi lääkintälaitteiden ohjelmistojen vaatimustenmukaisuutta tutkivassa SAHCE-projektissa (Pöyhönen et al. 2002). UL ilmaisee toimintansa kotisivuillaan seuraavasti:

UL sertifioi eräiden teollisuusalojen ohjelmoitavia järjestelmiä, yhtenä kohdealueena ohjelmistoilla ohjattavat lääkintälaitteet Programmable Electrical Medical Systems (PEMS). PEMSit kattavat sulautetut mikroprosessoripohjaiset ohjelmistot, joiden häiriötoiminta voi aiheuttaa henkilövahinkoja. UL:n toimiala on tämän mukaan henkilöturvallisuus. Sertifioitaviin ohjelmistoihin kuuluvat tietokoneohjelmat, toimintaohjeet ja turvallisuuteen liittyvien toimintojen käyttämät tiedot. UL-sertifiointi käsittää myös ohjelmiston kehitysprosessin auditoinnin.

Lääkintälaitteiden ohjelmistoja ovat mm. diagnosointiohjelmistot, hoitoon liittyvät ohjelmistot ja muut turvallisuuteen liittyvät toiminnot. Lääkintälaitteisiin kuuluvat verianalysaattorit, kuvantamislaitteet, valvontalaitteet ja muut laboratoriolaitteet.

UL myöntää onnistuneen ohjelmistoarvioinnin tuloksena sertifikaatin, jossa mainitaan että ohjelmisto on tarkastuksessa noudatettavien arviointikriteerien mukainen. Arviointikriteerien ja vaatimusten tärkein lähde on UL (1998) "Standard for Software in Programmable Components", jota kehitettiin yhteistyössä alan teollisuuden ja viranomaisten kassa. UL:n turvallisuustarkastus voi asiakkaan toivomuksista riippuen sisältää useita kansainvälisiä standardeita, mm. EN-IEC 61508 ja ANSI/ISA S84.01.

Etenkin Euroopassa on totuttu siihen, että riippumattomat yritykset arvioivat kehitysprosessien SFS-EN ISO 9001 standardin vaatimustenmukaisuuden. Standardi SFS-EN ISO 9000-3 edellyttää mm. dokumentoitua laadunjohtamisjärjestelmää.

UL:n ohjelmistoarvioinnit alkavat mahdollisimman varhaisessa elinkaarivaiheessa ja jatkuvat koko kehityksen ajan. Mahdollisilla varhaisilla epäkohtiin puuttumisilla halutaan välttää ohjelmiston laajaa uudelleensuunnittelua ja siten painottaa ohjelmistokehitystä sekä ajallisesti että kustannusten osalta oikeisiin kohteisiin.

Sertifioinnin laajuus riippuu tarkasteltavan tuotteen suuruudesta, kompleksisuudesta ja toiminnallisuudesta sekä käytettävissä olevasta dokumentaatiosta (valmiustaso, tarkkuus). Sertifiointi sisältää kaksi osaa, joista ensimmäinen on tuotedokumentoinnin turvallisuusosien arviointi, toinen kehityksen elinkaaritoimien arviointi (riskianalyysit, suunnittelukatselmoinnit, testit ja muutosten hallinnat). Sertifiointi voi osin perustua asiakkaan omiin testauksiin tai teettämiin testauksiin. Perusohjelmistot katselmoidaan myös.

Joillakin organisaatioilla on omia riippumattomia sertifiointiosapuolia. NASAlla on Independent Verification & Validation Facility (http://www.ivv.nasa.gov/). NASA IV&V suorittaa prosesseihin ja tuotteisiin kohdistuvat verifioinnit tarkistamalla, testaamalla, analysoimalla ja demonstroimalla. Riippumattomuuden määrittely on usein tuottanut ongelmia (esim. EN-IEC 61508.n tuottamissa SP 65:n WP 9:ssä ja WP10:ssä). NASA on ratkaissut ongelman määrittelemällä teknisen riippuvuuden eri tavalla kuin hallinnon riippuvuuden. Ensin mainitussa tekninen IV&V-henkilöstö on erillään ohjelmaa tuottavasta ryhmästä, toisessa eli hallinnon riippumattomuudessa NASA IV&V:lle on uskottu paljon valtaa erillisenä NASA-tuotteen toteuttamiseen liittyvistä vastuista, mm. kehityssopimukset. Kehitysryhmät voivat olla myös NASAn ulkopuolisia sopimusasiakkaita. NASA IV&V Facilityn toimintakohteista saa lisätietoa internetistä saatavilla olevasta julkaisusta http://www.ivv.nasa.gov/about/IVV_Rpt.pdf.

Myös Euroopan avaruusjärjestö ESA käyttää riippumattomia verifioijia ja validoijia sekä luotettavuuden analysoijia (mm. VTT Automaatio). Oleellista näiden kaikkien riippumattomien osapuolten (UL, NASA IV&V Facility, ESA:n IV&V:t) kohdalla on kuitenkin se, että ne eivät esitteiden mukaan taloudellisesti vastaa siitä, että ohjelmisto ei olekaan sitä laatua tai luotettavuutta kuin he ovat antaneet arvioinnissaan ymmärtää. Ne esittävät mielipiteitä ohjelman luotettavasta toiminnasta asiantuntijalausuntoihin vedoten sekä formaaleja todistuksia standardien ja määräysten vaatimustenmukaisuudesta. Sama pätee muuhunkin sertifiointitoimintaan. Sertifiointiyritys on vastuussa vasta sitten, jos se on tehnyt vakavia virheitä arviointiprosessissaan. Sertifioivat yritykset antava lisävakuuksia valmistajalle, mutta varsinaisen epäonnen seuraukset kärsivät muut osapuolet. Ohjelmistoteollisuus ei ole valmis tässä mielessä ”vahvoihin” takuisiin.

Alussa mainitun National Security Computer Associationin, NSCA, lähestymistyyli poikkeaa edellisistä mielenkiintoisella tavalla. Yritys sertifioi eli antaa takeita sille, että tietyssä sovelluksessa ei ole tiettyjä tunnettuja ongelmia. Ryhmä asiantuntijoita kokoontuu säännöllisesti päättämään, mitkä tunnetut ongelmat otetaan tarkastettavaksi. NSCA myös luettelee ne ongelmat, joihin he eivät voi puuttua. Lähestymistyyli on huomattavasti helpompi kuin yrittää taata ohjelmiston luotettavuus tietylle todennäköisyydelle.

Muilla kuin ohjelmistotekniikan aloilla sertifiointilaitokset tekevät lähinnä standardien vaatimustenmukaisuuden osoittamisia sekä prosessille että tuotteelle. Useilla toimialoilla (lääketiede, ilmailu ja ydinvoimala) on valmiit standardit tätä varten. Ohjelmistotekniikan alalla ei ole koskaan ollut vastaavan tasoisia standardeita.

6.3 Testaukset ja analyysit takuun välikappaleena

Kuten alussa todettiin, testien kattavuus on yksi merkittävimpiä luotettavuustekijöitä. Myös käytännöllisimmät ohjelmiston luotettavuuden arviointimenetelmät perustuvat testauksiin. Niistä on osoituksena mm. viitteiden Musa et al. (1987) ja Lyu (1996) esittämät luotettavuusmallit. Riittävätkö testitulokset ja -arvioinnit takuun pohjaksi? Useissa lähdeteoksissa (mm. Kaner & Pels 1998, Musa 1998) esitetään näkemyksiä, ettei ohjelmaa kyetä täysin testaamaan kaikkien virheiden löytämiseksi.

Merkittävien virheiden löytämisessä on toisenlaisiakin mielipiteitä kuin edellä mainitut. Kaikkia virheitä ei ehkä kyetä löytämään, mutta tulosta kyetään parantamaan yksinkertaisille ohjelmistoille ja keskittämällä testaamista kriittisimpiin osiin. Käytännössä on ollut ohjelmia, joista asiakkaat ovat raportoineet vain muutaman merkityksettömän virheen. Erityisesti turvallisuuteen liittyvät ohjelmistopohjaiset suojausjärjestelmät ovat osoittautuneet tässä mielessä virheettömiksi. Tosin niidenkin laadunvarmistuksessa on

muitakin menettelyitä kuin testaamiset (mm. laatujärjestelmä, monipuoliset verifioinnit ja validoinnit tarkastusmenettelyineen ja analyyseineen).

Testausten kattavuus yhdessä muiden verifiointimenetelmien kanssa onkin ratkaisevassa asemassa ohjelmiston luotettavuuden kannalta. Riskienhallintaan ja riskianalyyseihin kiinnitetään joissakin yrityksissä erityistä huomiota, mutta useimmissa suhtautuminen niihin on ponnetonta. Asiakkaan raportoimat tiedot virheistä saattavat olla valmistajalle jo ennestään tuttuja jo senkin takia, että ei ole kustannussyistä haluttu korjata kaikkia havaittuja virheitä ennen julkistamista. On ehkä oletettu, ettei virhe ole niin merkittävä, että asiakas raportoi siitä tai ei havaitse virheeseen liittyvää virhetoimintoa.

Ohjelmat eivät välttämättä suoriudu kaikissa tietokoneissa, joihin niitä on myyty. Valmistajat eivät ole silloin katsoneet taloudelliseksi testata kattavasti konfiguraatiota[9]. Konfiguroinnin ongelmat eivät kaikki kuitenkaan ole valmistajan vikoja esimerkiksi tapauksissa, joissa myynnin jälkeen on vaihdettu liittyvä ohjelma toiseen tai uuteen versioon.

Testausten riittävyyttä ovat myös tutkineet Voas & Payne (2000) projektissaan, jonka tuloksena kehitettyyn testimenetelmään tutustutaan seuraavassa luvussa lähemmin. Heidän menetelmässään pureudutaan ohjelmistovirheen etenemiseen näkyväksi virhetoiminnaksi ja testeihin, joilla eteneminen kyetään havaitsemaan. Luvussa 6.3.2 tarkastellaan Musan (1998) lähestymistyyliä ja luvussa 6.3.3 lähteen (Harju 2000) esittämää riskienhallinnan lähestymistyyliä.

6.3.1 Luotettavuuspisteytys

Voas ja Miller (1995) kehittivät luotettavuuspisteisiin perustuvan mittausvälineen[10]. He ovat määritelleet käsitteen ”dependability score”, jolla tarkoitetaan ”luotettavuuspisteitä”. Ne eivät ole absoluuttisia luotettavuuden mittareita, vaan niiden avulla vertaamalla valitaan ohjelmistoja (ja komponentteja). Menettelyä käyttävät ohjelmistovalmistajat yrittävät vakuuttaa asiakkaat korkeilla ja perustelluilla pisteillään.

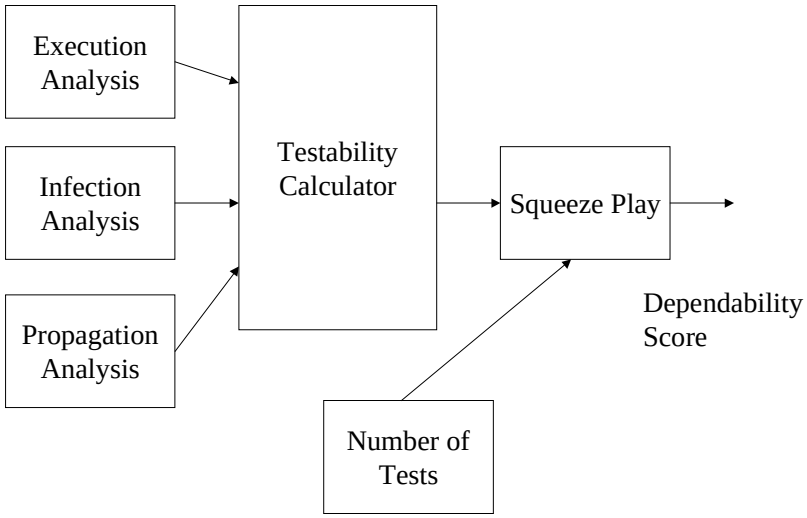
Menetelmä perustuu kattavaan testaamiseen. Pisteet kasvavat sitä enemmän mitä järjestelmällisin lisätestein pystytään paljastamaan tietyn tyyppisiä virheitä. Tietyt testityypit tuottavat pisteitä enemmän kuin toiset. Heidän menetelmässä ei vain lasketa testitapausten määrää vaan myös painotetaan vikojen paljastumista.

---

9 Konfiguraatiotesteissä testataan ohjelma mm. muiden ohjelmien, eri tyyppisten tietokoneiden ja eri käyttöjärjestelmäversioiden kanssa.

10 Menetelmää on edelleen sovellettu ohjelmistokomponenteille (Voas ja Payne 2000).

Luotettavuuspisteityksen prosessia esittää kuva 15. Prosessi muistuttaa tavallisia ohjelmiston luotettavuuden arviointimalleja (mm. Musa 1998), mutta siinä painotetaan testien merkittävyyttä. Tässä menetelmää tarkastellaan lähinnä sen innovatiivisuuden takia, sillä kovin paljon referenssejä menetelmän käyttökelpoisuudesta ei ole saatavilla.

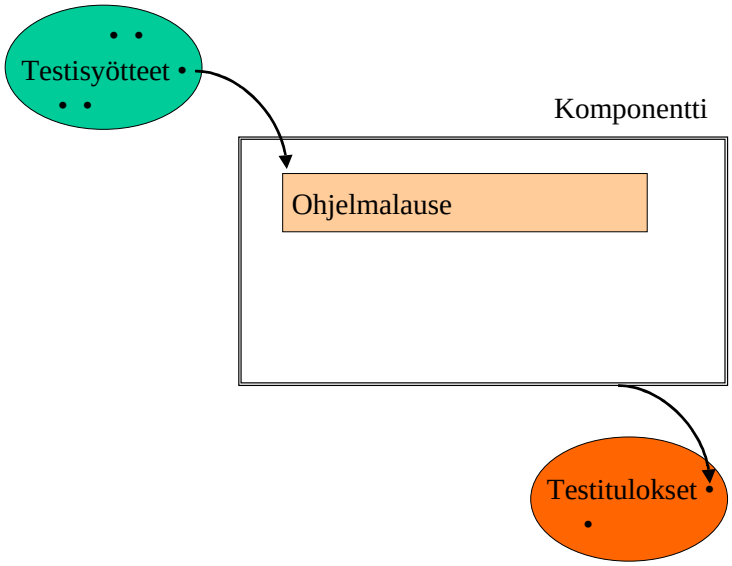


Kuva 15. Luotettavuuspisteytysprosessin periaate (Voas & Miller 1995).

Luotettavuuspisteytys koostuu *kolmesta analyysista*, niistä lasketusta *testattavuudesta*, sekä *pisteytyksestä*, joka puristetaan/squeeze testien lukumäärästä ja testattavuudesta. Lopputuloksena on luku, joka kertoo komponentin suhteellisen luotettavuuden. Analyysit ovat kirjallisuudesta tuttuja:

1. Suoritusanalyysi
2. Virheen arvaus
3. Etenemisanalyysi

Suoritusanalyysilla (Execution Analysis) selvitetään millä todennäköisyydellä tietty ohjelmalause joutuu suoritettavaksi. Analyysin taustalla ovat käsitteet ohjelmavirheestä (fault), virhetilanteesta (error) ja virhetoiminnosta (failure), joiden mukaan virhe eli bug on ohjelman sisäinen virhetilanne sijaitessaan jossakin koodilauseessa, ja virhetilanne johtaa virhetoimintaan eli ulkoisesti havaittavaan vikaantumiseen, kun koodilause suoritetaan. Virhetoimintaa ei siis tapahdu, jos virheen sisältämää lausetta ei suoriteta.



Kuva 16. Millä todennäköisyydellä ohjelmalause suoritetaan?

Dynaamista virheen arvausta (Error Guessing, Infection/Injection Analysis) on sovellettu osaksi elektroniikan ja ohjelmiston validointia (Clark & Pradhan 1995, Benso et al. 1999). Tavoitteena on selvittää millä todennäköisyydellä syötetty virhetilanne (virhe koodilauseessa, ks. kuva 16) johtaa/saastuttaa komponentin virhetoiminnoksi. Virheen arvausmenetelmä on ollut pinnalla jo vuosia. Sen heikkoutena on ollut geneerisyyden puute, sillä menetelmät ovat tietokoneista ja sovelluksista riippuvia. Virheen arvauksessa lähdekoodiin koodataan muunnos eli mutantti. Mutantit ovat olleet tutkimuksen alaisena jo pitkään, mutta vieläkään ei tunnisteta, minkä tyyppisillä mutanteilla kyetään jäljittämään ohjelmointivirheitä.

Dynaamisella virheen etenemisanalyysilla haetaan todennäköisyyttä sille, että virhetilanne etenee toiseksi virhetilanteeksi. Tietty virhe koodilauseessa johtaa esimerkiksi toisen käskyn väärään suoritukseen. Etenemisanalyysilla arvioidaan tiettyjä ohjelmiston ominaisuuksia, joita muilla keinoilla on vaikea arvioida. Ominaisuuksia ovat mm. testattavuus, turvallisuus ja tietoturva. Dynaamisella analyysilla joudutaan ohjelma suorittamaan yhä uudelleen ja uudelleen, mikä on työlästä ja kallista.

Edellä kuvatuista analyyseista saadut todennäköisyysluvut kerrotaan keskenään ja saadaan uusi todennäköisyysluku, joka esittää ennustetta siitä, millä todennäköisyydellä tietyssä testattavassa lauseessa on ohjelmistovirhe. Testattavuudella tarkoitetaan tässä yhteydessä mittaa siitä, miten monta testiä tarvitaan, jotta tietty todennäköisyys pystytään saavuttamaan.

Testattavuuden muodostamista edellä kuvatulla tavalla perustellaan sillä, että ohjelmistovirheen johtaminen näkyväksi virhetoiminnoksi edellyttää seuraavia todennäköisyystapahtumia:

1. Todennäköisyys sille, että virhe syntyy.
2. Ehdollinen todennäköisyys sille, että virhekohta suoritetaan, kun virhe on olemassa.
3. Ehdollinen todennäköisyys sille, että virheellinen data johtaa virheelliseen tulokseen, kun virhekohta on suoritettu.

Viitteessä (Voas & Payne 2000) esitetään tarvittavat menettelytavat luotettavuuspisteiden ja testimäärien laskemiseksi. Testimäärät N saadaan yhtälöstä:

$$N = \frac{\ln(1 - D_\alpha)}{\ln(1 - t_\alpha)}, \tag{1}$$

missä luotettavuuspiste D_α saadaan yhtälöstä:

$$D_\alpha = 1 - (1 - t_\alpha)^N, \tag{2}$$

missä testattavuuspiste t_α saadaan yhtälöstä:

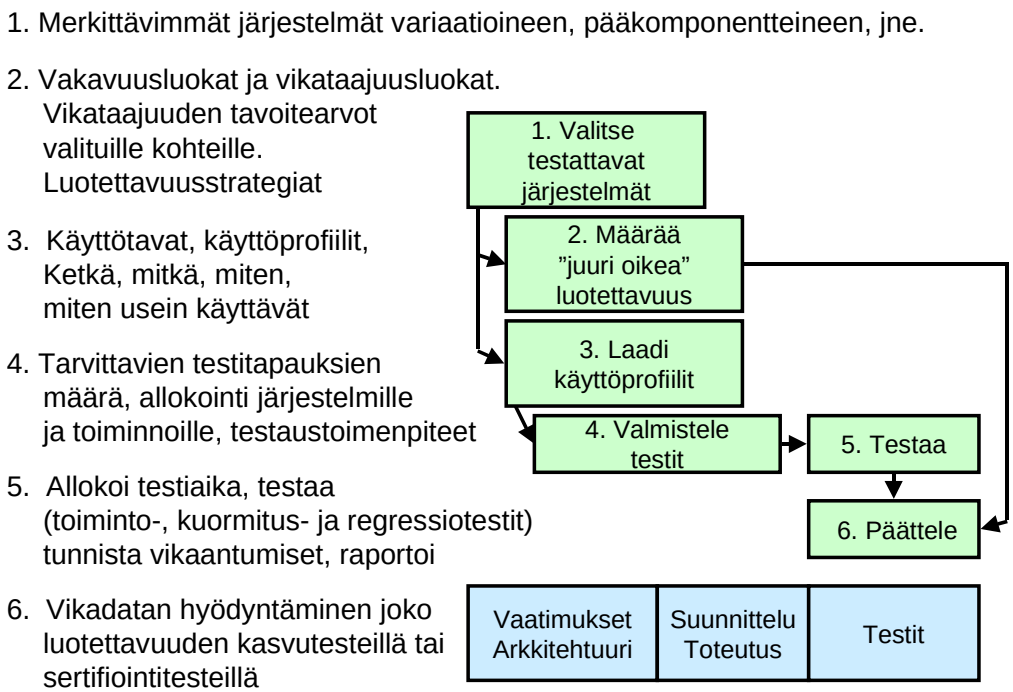
$$t_\alpha = E_\alpha I_\alpha P_\alpha, \tag{3}$$

missä E_α, I_α ja P_α ovat suoritusestimaatti, injektioestimaatti ja etenemisestimaatti ohjelmistokomponentille α vastaavien analyysien mukaan. Luotettavuuspiste D_α ei ole tavallinen metriikka luotettavuuden arvioimiseksi tai vikaantumistodennäköisyyden määrittämiseksi, vaan se vertaa suoritettuja testimääriä virheiden ennustelukuihin.

6.3.2 Musan ohjelmiston luotettavuustekniikka

Musa (1998) kutsuu kehittämäänsä menetelmää standardiksi ohjelmiston luotettavuuden arviointitekniikaksi. Menetelmän keskeiset osaprosessit (ks. kuva 17) sisältävät tarkoin määriteltyjä osatehtäviä, sääntöjä ja laskentamalleja, joita ei tässä yhteydessä ole asiayhteyden takia syytä käydä täsmällisesti läpi. Tässä esitetään lyhyt katsaus Musan menetelmään, joka erittelee tarkastelupainotuksissa erittäin kriittiset, vähemmän kriittiset ja muut toiminnot. Menetelmä on tarkoitettu siten kaikenlaisille järjestelmille, sekä laa-

joille ja kompleksisille että esimerkiksi kriittisiä suojaustoimintoja sisältäville ohjelmistopohjaisille järjestelmille.



Kuva 17. Musan (1998) ohjelmiston luotettavuustekniikkaa.

Menetelmä pystyy noudattamaan mitä tahansa ohjelmiston kehitysprosessia, esimerkiksi vesiputousmallia. Tarkastelu alkaa mahdollisimman varhaisessa vaiheessa elinkaarta, mieluimmin jo laadittaessa konseptia tai määriteltäessä käyttäjävaatimuksia. Tarkastelu yksinkertaistuu mitä nopeammin kyetään valintoja tekemään (*kuva 17*: kohta 1).

Juuri oikean luotettavuuden (so. toimintavarmuuden Musan mukaan) (kuva 17: kohta 2) määrittäminen on välttämätöntä ohjelmiston luotettavuustekniikan hyväksikäyttämisessä. Määrittäminen tapahtuu viidessä vaiheessa:

1. Määrää tuotteen virhetoiminnot vakavuusluokittain.
2. Valitse yhteinen mitta kaikille asiaankuuluville järjestelmille.
3. Aseta testattaville tuotteille vikaantumistiheyden tavoitearvot.
4. Laske ohjelmistolle vikaantumistiheyden tavoitearvot.
5. Suunnittele tavoitearvot täyttävä ohjelmisto.

Virhetoimintojen tarkastelussa selvitetään minkälaisia seurauksia tuotteen toiminnan epäonnistuminen voi aiheuttaa. Virhetoiminnot luokitellaan sopivaan määrään vakavuusluokkia, joista yhden muodostavat tapaukset, joita ei sallita lainkaan, ja toisen luokan tapaukset, jotka ovat lähinnä kosmeettisia. Muut luokat ovat sopivasti näiden äärilaitojen välillä. Virhetoiminta määritellään käyttäjälähtöisesti siten, että se on poikkeama käyttäjävaatimuksista. Virhe on suunnittelijalähtöinen. Se syntyy jossakin vaiheessa kehittämistä ja seurauksena on ulospäin näkyvä virhetoiminto.

Kaikilla samaan vakavuusluokkaan kuuluvilla virhetoiminnoilla on vastaavansuuruinen seurausvaikutus käyttäjälle. Vaikutukset voidaan ilmaista taloudellisina menetyksinä, epäkäytettävyytenä tai jonakin muuna luotettavuusattribuuttina (ks. taulukko 5 ja taulukko).

Taulukko 5. Kustannuksiin perustuva virhetoimintojen vakavuusluokitus.

Vakavuusluokka	Suuruus (€)
1	> 100 000
2	10 000–100 000
3	1000–10 000
4	< 1000

Taulukko 6. Käytettävyyteen perustuva virhetoimintojen vakavuusluokitus.

Vakavuusluokka	Suuruus
1	Perustoiminnon epäkäytettävyys
2	Merkittävän toiminnon epäkäytettävyys
3	Toimintojen epäkäytettävyys, mikä ei estä työskentelyä
4	Toimintojen merkityksetöntä vajavuutta

Vikaantumistiheys on vaihtoehtoinen ilmaisutapa luotettavuudelle. Siinä yksinkertaisesti esitetään, miten monta vikaa esiintyy tiettynä ajankohtana (esimerkiksi 1 virhetoiminto 1000 tunnissa). Muuta yksinkertaista ei siinä sitten olekaan. Ohjelmistolle suoritusaika on oleellinen aikatyyppi, koska jos ohjelmisto ei ole suorituksessa, se ei myöskään aiheuta virhetoimintoja toisin kuin elektroniset tai mekaaniset osat. Suoritusaikaakin merkittävämpi mitta olisi suorituksessa olevien käskyjen lukumäärä, mutta lukumäärän määrittäminen ei ole niin helppoa kuin suoritusajan määrääminen. Suoritusajan valitsemista puoltavat myös yhtäläiset valmiudet laitteiston luotettavuusarviointiin, sillä perimmäisenä tavoitteena on aina laskea järjestelmän luotettavuus, ei niinkään pelkästään ohjelmiston (Musa 1998).

Vikaantumistiheyden tavoitearvo asetetaan kokonaisjärjestelmälle ottamalla huomioon käyttäjän ja asiakkaan erityistarpeet ja -odotukset sekä niiden suhde järjestelmän ominaisuuksiin. Järjestelmän laatuominaisuuksista monet, kuten kehitysaika ja -kustannukset, ovat ristiriidassa vikaantumistiheyden tavoitearvon kanssa. Pyrkimys esimerkiksi pienentää kehitysaikaa tai -kustannuksia johtaa kasvavaan vikaantumistiheyteen. Toiminnallisuuden vähentäminen merkitsee kaikkien kolmen ominaisuuden vähentymistä, mutta kasvattaminen voi alentaa käyttökustannuksia niin, että asiakkaalle kohdistuvat kokonaiskustannukset investoinnista ja käytöstä ovat edullisemmat.

Edellä mainittujen ominaisuuksien (vikaantumistiheys, kehitysaika, -kustannukset ja toiminnallisuus) tarkan suhteen määrittäminen edesauttaisi ristiriitatilanteiden selvittämistä ja vikaantumistiheyden tavoitearvon asettamista. Monet vikaantumistiheyteen vaikuttavat tekijät ovat sikäli tunnettuja, että tiedetään niiden arvojen nostamisen tai vähentämisen vastaavasti nostavan tai vähentävän vikaantumistiheyttä. Tarkat suhteet ja vaikutusarvot eivät ole tiedossa. Niiden hankkimiseksi tarvittaisiin datan keruuta ja analyysia isosta joukosta vastaavia kehitysprojekteja.

Vikaantumistiheyden tavoitearvon määrittäminen ei ole kovin helppoa (Musa 1998). Siksi se kannattaa tehdä keskiarvona kaikille tehtäville mieluimmin kuin yrittää raskasta määrittelyä yksittäisille, vaikkakin vain kriittisille operaatioille.

Kehitettävän tuotteen ohjelmistolle tulisi myös asettaa vikaantumistiheyden tavoitearvo, jolla on toimittajan antamissa takuutodistuksissa keskeinen sija. Tämä tapahtuu arvioimalla ensin odotettu vikaantumistiheys koko tuotteelle ja sitten laitteistolle. Jäljelle jäisi ohjelmiston tarvitsema osuus. Estimaattia täsmennetään käyttökokemuksista ja asiantuntijoiden kokemuksista siten, että takuuarvo saadaan täsmennettyä.

Suunnittelustrategioita vikaantumistiheyden tavoitearvon saavuttamiseksi ovat perinteiset menetelmät: virhevälttöisyys, virhesietoisuus ja virheen poistaminen ensisijassa katselmuksin ja testein. Musa (1998) toteaa, että näillä menetelmillä saavutetun vikaantumistiheyden aleneminen kyetään selvittämään vain approksimatiivisesti.

Käyttöprofiili (kuva 17: kohta 3) on joukko toimintoja ja niiden esiintymistodennäköisyyksiä. Käyttöprofiilista ilmenee mm. käyttäjät, ulkoiset järjestelmät ja systeemiohjaukset. Se laaditaan luettelemalla toiminnot (systeemivaatimuksista, prototyypeistä, käyttömanuaalin versioista). Näille toiminnoille määrätään esiintymistodennäköisyydet tai vikataajuudet, mitkä on suhteellisen helppo arvioida mm. kenttätietojen pohjalta.

Testitapausten valitsemiseksi valitaan käyttöprofiilit. Niiden avulla tai niiden esittämässä suhteessa kaikki toiminnot testataan. Esimerkiksi, jos tietyn toiminnon osuus on 15 %, testataan kokonaisuudesta 15 %.

Testien valmistelussa (kuva 17: kohta 4) Musa (1998) erottaa kaksi toimenpidettä: testitapausten ja testiproseduurien valmistelun. Periaatteessa etenkin turvallisuuteen liittyville järjestelmille tulisi kyetä laatimaan testisuunnitelmat lähtien kaikista tarvittavista syötemuuttujista. Turvallisuuteen liittyvien järjestelmien kriittisyys ja yksinkertaisuus korvaavat käyttöprofiilien ja testisuunnittelun joskus kalliiksi käyvät menettelyt.

Testejä ovat toimintotestit (tai muut erityistestit), kuormitustestit ja regressiotestit. Erityistestaamisella selvitetään ohjelmiston erityisominaisuudet ja -toiminnot erityiskohdittain ottamatta huomioon niiden välisiä riippuvuuksia. Kuormitustestauksessa selvitetään kaikki testitapaukset kerralla ottamalla huomioon vuorovaikutukset ja ympäristövaikutukset niin täydellisinä kuin mahdollista. Regressiotestit testataan, kun merkittäviä muutoksia on tehty, ja erityisesti silloin, kun kriittisiä toimintoja on muutettu. Testitapausten valmisteluun kuuluvat seuraavat vaiheet:

1. Arvioidaan tarvittavien testitapausten kokonaismäärä.
2. Jaetaan testitapaukset kaikkien merkittävien järjestelmien kanssa.
3. Jaetaan kunkin järjestelmän testitapaukset toiminnoille.
4. Spesifioidaan testitapaukset.

Uusien testitapausten kokonaismäärä arvioidaan käytettävissä olevan tai tarvittavan ajan ja henkilöstön suhteen ottamalla huomioon sekä kustannukset että määritelty vikaantumistiheyden tavoitearvo. Uudet versioinnit lisäävät testaustarvetta niin että siirtyminen automatisoituun testitapausten generointiin saattaa tulla kannattavaksi. Myös kuormitustesteissä käydään läpi useita samoja testitapauksia satunnaisvalitsemalla kenttää kuvaavin muuttuja-arvoja.

Allokoitaessa uusia testitapauksia eri kohdejärjestelmille, esimerkiksi sovellustuotteelle ja käyttöjärjestelmälle, painotetaan valintaa riskillä ja kohdejärjestelmän koolla. Testitapaukset allokoidaan järjestelmän eri toiminnoille tunnistamalla ensin kaikkein kriittisimmät toiminnot ja päättämällä niiden testaustarpeesta. Seuraavaksi käsitellään kaikki muut toiminnot määrittämällä niille allokoidut todennäköisyydet.

Testisuoritukset (kuva 17: kohta 5) koostuvat testiajan allokoinnista, testitapahtumasta ja virhetoimintojen tunnistamisesta. Käytettävissä oleva testiaika jaetaan järjestelmittäin testaustavoille ja toiminnoille sen mukaan mitä luotettavuusteknistä testaustapaa (ks. kohta 6 alla) tarvitaan. Niistä luotettavuuden kasvutesteissä käytetään kaikkia kolmea testaustapaa, toiminto-, regressio- ja kuormitustestejä, riippumatta siitä, seuraako niitä sertifiointitestit. Sertifiointitesteissä koko testiaika kohdistetaan kuormitustesteille. Musan mallin mukainen testaaminen voidaan aloittaa vasta sen jälkeen kun valittu järjes-

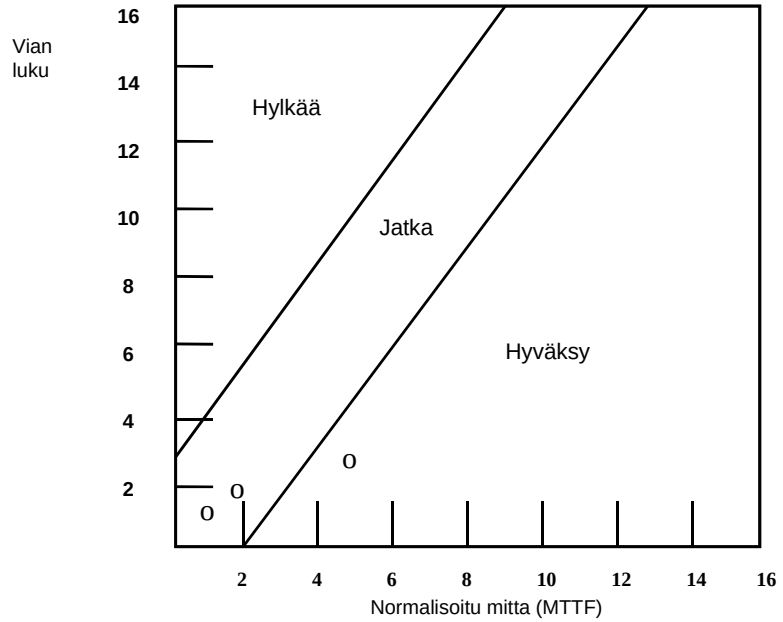
telmä on integroitu siten, että toiminnot ovat suorituskuntoisia sekä läpikäyneet verifiointiprosessin testeineen.

Testeistä saatua vikadataa hyödynnetään Musan menettelyssä useissa päätöstilanteissa (kuva 17: kohta 6), joita ovat mm. kronologisessa järjestyksessä esitettynä seuraavat:

1. Ohjelmistokomponenttien hyväksyminen tai hylkääminen
2. Ohjelmiston kehitysprosessin hallitseminen
3. Järjestelmän hyväksyminen tai hylkääminen
4. Tuotteen julkistaminen

Hyväksymisen ja hylkäämisen tukena ovat sertifiointitestit (eli siis kuormitustestit), kun taas luotettavuuden kasvutesteihin tukeudutaan kehitysprosessin eri vaiheissa ja tuotteen julkistamisessa.

Sertifiointitestien yleistä kulkua ja merkitystä voidaan selittää kuvan 18 esittämällä demonstrointikartalla menemättä yksityiskohtiin, joita löytyy viitteestä Musa (1998: kohdat 6.1 ja 6.3.3). Kuvassa ovat alueet sekä hyväksyntää, hylkäystä että testien jatkamista varten. Jakaminen näihin alueisiin riippuu sertifiointitestaamiselle asetetuista tavoitteista ja riskinoton näkökulmista. Aina uuden vian löytyessä tarkastellaan, onko testaaminen siirtymässä kohti hyväksyttävää vai hylkäävää aluetta. Musan (1998) mukaan tällä tavoin menetellen saadaan mahdollisimman nopeasti selville testausten riittävyys. Musa ohjeistaa sekä aluerajojen laadintaa, testaamista komponenttitasolla ja järjestelmätasolla.



Kuva 18. Luotettavuuden demonstroiminen (Musa 1998).

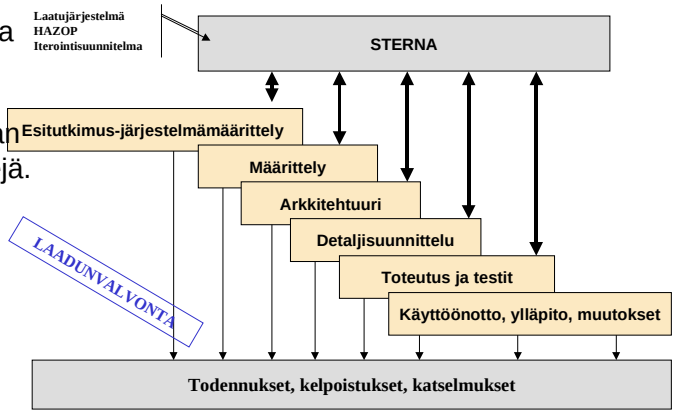
Sertifiointitestien päämääränä on joko hyväksyä tai hylätä. Ne on tarkoitettu myös ulkopuolisille ohjelmistoille ja ulkopuolisten tekemiksi, kun taas luotettavuuden kasvutestit suuntautuvat lähinnä valmistajan omaan tuotantoon ja sisäiseen käyttöön. Niillä selvitetään onko ohjelmisto riittävän luotettava julkistamista varten. Sertifiointitesteissä ei virheiden korjaaminen ole oleellista, mutta luotettavuuden kasvutesteissä havaitut virheet korjataan ja luotettavuutta arvioidaan sen jälkeen uudelleen.

6.3.3 Tiira – riskienhallinnan menetelmä

Tekesin teknologiaohjelman ETX eräässä projektissa kehitettiin Tiira (t. Stérna) -niminen ohjelmiston luotettavuusvaatimusten validointimenetelmä (Harju 2000). Menetelmä sekä ohjeistaa ohjelmistokehittämisen prosesseja että arvioi vikaantumisen eli virhetoimintojen esiintymistiheyttä projektin edetessä vaihe vaiheelta. Tiira ei liity pelkästään turvallisuuteen, vaan myös muut luotettavuusattribuutit ovat tarkastelun alla. Ohjaaminen tapahtuu nk. luotettavuusprofiilin avulla (ks. kuva 3), jota on em. viitteessä selostettu lähemmin.

Tiiran (Stérnan) syöksy pääpiirtein:

1. Määrätään riskitaso toiminnoille järjestelmätasolla tai ohjelmiston toiminnallisessa määrittelyssä.
2. Laaditaan luotettavuusprofiili (R2 A3 M1 S1), allokoidaan sitä vaiheissa.
3. Arvioidaan riskinvähennyksen suuruus (vikasietoisuus, -välttöisyys, jne.)
4. Tarkistetaan vaihetuotteiden oikeellisuus ja täydellisyys (iteratiivinen prosessi).
5. Tarkistetaan avainlauseilla välituotteet.
6. Suositetaan ja ohjeistetaan kehitysprosesseja ja testejä.
7. Arkkitehtuurivaihe luotettavuuden kannalta tärkeä:
   Riski voi kasvaa



Kuva 19. Tiira kohdentaa ohjelmiston kehittämistoimintaa siten, että voidaan vakuuttautua siitä, että kriittisiä virhetoimintoja ei ole (Harju 2000).

Keskeisintä menetelmässä on selvittää vaiheittain todellinen virhetoimintojen esiintymistiheys ja haluttu tavoitearvo tietylle riskitoiminnalle. Riskienvähentämisen periaatetta noudatetaan monissa muissakin menettelyissä, mm. kattostandardissa EN-IEC 61508, ja lähinnä autoteollisuudessa hyödynnetyssä RPN[11]-menettelyssä (mm. Palady 1998).

Menetelmä perustuu sekä induktiivisiin että deduktiivisiin päättelytekniikoihin, jotka ovat tuttuja laitteistojen analysoinneista ja joita on onnistuneesti sovellettu ohjelmistolle aivan viime vuosina. Suurimpana soveltamisongelmana ohjelmistoille on oikean lähtötason ja tekniikoiden sekä kattavien virhetoimintojen vallinnat.

Tarkastelun lähtötason valinta riippuu monista tekijöistä, mm. saatavilla olevasta *informaatiosta, resursseista ja yhteensopivuusvaatimuksista* muun toiminnan kanssa. *Informaatiota* tarvitaan luotettavuuden tarkastelun tavoitteista. Sitä on ensi sijassa luotettavuusarvio ohjelmiston virheettömyydestä, mikä johtaa luotettavuusattribuuttien tavoiteasetteluun sekä viittaa informaatioon mm. tekniikoilla tunnistettavista virhetyypeistä. Halutaan myös parantaa yrityksen omaa ohjelmistotuotantoa esimerkiksi kehittämällä todennus- ja kelpoistusprosessia, laadunvarmistusta tai dokumentointia. Lisäksi luotettavuuden analysointimenetelmä voidaan liittää osaksi ohjelmistotuotantoa mm. suosittelemalla testitapauksia yms.

Resursseilla tuetaan tarvittavan informaation tuottamista. Siihen sisältyvät tekniikan vaatimat ohjelmistotuotannon dokumentit sekä asiantuntija- ja muun henkilöstön tarve. Tarvittava asiantuntemus riippuu tekniikan käytön helppoudesta, työkalujen saannista ja muista vastaavista ominaisuuksista. Arviointi helppokäyttöisyydestä ei ole kovin yksinkertaista. Ohjelmiston vikapuuanalysointi vaatii kvantitatiivisesti perehtymistä ja kouluttautumista aiheeseen. Myös mahdollisten virheiden tunnistaminen vaatii harjaannusta. Tekniikkojen järjestelmällisyys tarkoittaa usein myös sen työläyttä, mutta erityisesti vain niissä soveltamistapauksissa, joissa järjestelmällisyydestä ei päästä eroon tiukasti kohdentamalla tarkastelua vain kaikkein kriittisimmille ja samalla suppeille aloille.

Tekniikat voidaan myös suorittaa eri tarkkuustasoilla riippuen tarkasteltavan kohteen laajuudesta, käytettävissä olevista resursseista sekä kohteen riskitasosta. Suorituksen tarkkuus jaetaan kolmeen analysointitapaan: epäformaali, hyvin suunniteltu ja formaali.

Yhteensopivuusominaisuus kuvaa ne luotettavuuden analysointitekniikoiden ominaisuudet, jotka tarvitaan tekniikan sovittamiseksi yrityksen ohjelmistotuotantoon. Nykykäytäntöä kuvaavat yritysten laatudokumentit ja erilaiset tuotantoon kohdistuvat säännöt ja

[11] Risk Priority Number

ohjeet, mm. ohjelmointisäännöt sekä tuotosdokumentit eli aineisto, jota analyyseissa tarvitaan.

Luotettavuuden analysointitekniikoiden tehokkuutta kuvataan myös niiden sopivuudella yhdistää analyysi yhteen tai useampaan elinkaaren vaiheeseen. Analysoinnit voidaan yleensä tehdä yhdessä tai useammassa tai mahdollisesti kaikissa elinkaaren vaiheissa.

Lisäksi riippumattomuus analysoinnin tekijöiden ja kehitystyön tekijöiden välillä vaikuttaa lähtötason ja analyysitekniikoiden valintaan. Riippumattomuus jaetaan EN-IEC 61508:ssa riippumattomuudeksi projektista, osastosta ja organisaatiosta.

Virhetoimintojen ja vioittumistapojen valinnat ovat onnistuneen tarkastelun perusedellytyksiä kaikissa vika-analyyseissa. Stérnassa hyödynnetään eräänlaisia tarkistuslistoja, nk. avainlauseita kaikissa ohjelmiston elinkaaren valituissa vaiheissa. Avainlauseet soveltuvat niin vioittumistapojen kuin niiden syiden, vaikutusten, havainto- ja kontrollimekanismien tunnistamiseen (kuva 19).

7. Johtopäätökset

Tässä julkaisussa käsiteltiin neljää erillistä teemaa: luotettavuustekijöitä, eheystasojen hyödyntämistä, COTSien ja niistä koostuvien järjestelmien luotettavuutta ja luotettavuustakuuita.

Eheystasojen hyödyntämisessä (ks. luku 4) tarkasteltiin mahdollisuuksia hyödyntää determinististä ja kvalitatiivista säännöstöä eri luotettavuus- ja turvallisuustasoilla. Menettelytyyli on käytössä erityisesti standardissa EN-IEC 61508 (2001), joka on suuntautunut turvallisuuteen liittyviin järjestelmiin (TLJ). Siinäkin monimutkaisuus on yksi tekijä, jonka mukaan tarkasteltava järjestelmä asetetaan tietylle turvallisuuden eheystasolle (TET). Mutta muitakin tekijöitä on. Tärkein niistä on tietysti itse ohjattavan kohteen turvallisuus. Monimutkaisuuden hallinnalla, suunnittelu ja mittaaminen mukaan lukien, on siten merkitystä turvallisuuden eheystason alentamisessa. TET määrittelee vaatimukset TLJ:lle koskien koko sen elinikää suunnittelijan työpöydältä käyttöön, muutoksiin ja purkuun.

Tulisi selvittää merkittävät monimutkaisuuteen vaikuttavat deterministiset säännöt ja ennen kaikkea niiden vaikutus osoittamismielessä. Siis olisi tutkittava, kuinka paljon niiden käyttäminen täsmällisesti vähentää riskiä tietyissä olosuhteissa ja ympäristössä (ohjattava kohde, järjestelmä, projekti ja yrityskulttuuri). Eheystasoihin tulisi myös liittää henkilöiden kyvykkyysvaatimukset. Henkilösertifiointi on perusteorian puuttumisen takia kuitenkin suhteellisen vaikeaa.

Standardit pyrkivät joskus liiankin helppoihin ratkaisuihin. EN-IEC 61508 luokittaa järjestelmät jatkuvatoimisiin säätö- ja ohjausjärjestelmiin sekä suojausjärjestelmiin. Näiden väliltä ei ole mitään jakoa. Helppoa luokitusratkaisua ei ole esimerkiksi verifiointien ja validointien kattavuuden määräämiselle. Diagnostiikan kattavuusmenetelmän kohdalla ei ole päästy selkeään tulkintaan siitä, mitä kattavuus voisi tarkoittaa yhdessä muiden virheen ilmaisukeinojen kanssa. Riittävän ja juuri oikean luotettavuuden osoittamismielessä kyse on kokonaismenetelmien kattavuudesta. Ei siis pelkästään analyysien ja testien. Suunnittelu- ja laatusäännöt olisi myös osattava ottaa huomioon eheystason mukaisesti.

Turvallisuuden eheystasojen määrittämisessä ei oteta riittävän implisiittisesti hyötynäkökohtia huomioon. Määrittämismenetelmissä tulisi hyötyjen esiintyä siinä kuin vaarojenkin. ALARP[12]-periaate on hyvä yleisperiaate arvioitaessa riskien vähentämisen kustannustehokkuutta, mutta käytännön ohjeita sen noudattamisesta TET:ien määrittämisessä ei ole. Hyötynäkökohdat tulisi ottaa huomioon jo turvallisuusanalysoinneissa.

[12] As Low As Reasonable Practicable

Eheystasomenettelyä tulisi kehittää niin, että järjestelmän sisällä tietyt osat voisivat olla alempaa eheystasoa kuin mitä järjestelmälle on periytynyt sen korkeimman toiminnon omaavan eheystason kautta. Silloin voidaan hyödyntää alemmankin tason valmisohjelmistoja ja COTS-ohjelmistokomponentteja, joita on tuotettu massamarkkinoille, mutta ei välttämättä tietylle korkealle eheystasolle.

Eheystason vaatimusten täyttyminen tulisi myös validoida riippumatta kehitysprosessin aikana tuotetusta informaatiosta. Arviointikriteerien avulla kehitystiimistä riippumaton osapuoli voisi arvioida, miten perusteellisesti kehitysprosessia on noudatettu ja tuotetta tarkistettu.

Riippumaton osapuoli voisi arvioida tuotteen kehitysprosessin onnistumista vertaamalla tarkistus- ja testituloksia tunnettuihin tietyn sovellusalan vikaluokituksiin. Vikojen luokittelu on kuitenkin osoittautunut hyvin vaikeaksi. Luokittelussa huomioon otettavia seikkoja oli aivan liian paljon ja niistä ei voida saada riittävän täsmällistä kenttätietoa. Osittain vaikeudet johtuvat siitä, että vikojen raportointiohjeet vikaluokittelumenettelyn osalta ovat olleet vajavaiset. Hieman täsmällisemmällä ohjeistolla virheiden taksonomiaa voitaisiin hyödyntää jo suunnittelunaikaisessa menetelmävalinnassa. Vikojen raportoinnista ja menetelmäsuositusten esittämisestä on kuitenkin aina hyötyä kyseiselle suositettavalle teollisuusalalle.

Turvallisuuden eheystasomenettely (TET-menettely) helpottaa suunnittelua, turvallisuuden eheyden arvioimista ja riskienhallintaa. Menettely yhtenäistää käytäntöjä, mistä hyötyvät suunnittelijoiden ja valmistajien lisäksi myös asiakkaat, loppukäyttäjät ja viranomaistahot. EN-IEC 61508:n ajattelutapa TET-menettelyssä onkin levinnyt hyvin laajalle. Se tunnetaan automaation turvallisuussuunnittelussa niin ydinvoimalaitos-, raideliikenne-, koneidenvalmistus- kuin prosessiteollisuusalalla. Uusia sektorikohtaisia standardeita on kehitteillä eri aloille.

Menettelytapa soveltuisi myös muille luotettavuusattribuuteille kuin turvallisuudelle. Käytännössä se merkitsisi erilaisten suunnittelusääntöjen ja -ratkaisujen sekä tarkastus-, katselmointi-, testaus- ja analyysimenetelmien sijoittamista tiettyyn luotettavuuden eheystasoon (dependability integrity level, DIL). Sijoittaminen edellyttäisi myös sekä teoreettisen että empiirisen luotettavuuskuvan saamista sijoitettavista menetelmistä ja tekniikoista. Kyse on myös niillä saavutettavan riskinvähennyksen suuruudesta. Läheisiä sukulaismenetelmiä DIL-menettelylle ovat erilaiset ohjelmiston kehitysprosessien luokitustavat, kypsyysmallit (CMM ja CMMI) ja kyvykkyyden osoittamismallit (mm. Spice). DIL-menettely keskittyisi kuitenkin vain luotettavuuteen: luotettavuuden suunnitteluun ja osoittamiseen.

Rautatieliikennettä koskeva standardi EN 50126 on kiinnostava siinä mielessä, että se käsittelee myös muita luotettavuusattribuutteja kuin turvallisuus. Siinä kuvataan luotettavuuden hallintaprosessi, luotettavuusvaatimusten määrittelyprosessi ja prosessi, jossa luotettavuusvaatimusten täyttyminen demonstroidaan. Luotettavuuden tavoitearvoja standardi ei kuitenkaan anna, eikä se keskity yksinomaan ohjelmistoihin, vaan mukana ovat ohjelmistoa sisältämättömätkin järjestelmät.

Lääkintälaitteiden valmistuksessa turvallisuus ja siihen läheisesti liittyvät käytettävyys eli palvelujen saatavuus, vaikuttavuus sekä suorituskyky ovat keskeisimmät viranomaistaholta tulevat laatuattribuutit. Valitettavasti sekä turvallisuus että käytettävyys ovat suoraan huonosti mitattavissa. Ohjelmistotekniikassa on kehitetty vuosien varalla kokonaisvaltaisia laadunmittaamistapoja, joita voidaan hyödyntää sekä prosessin ohjaamisessa että riittävän laadukkaan tuotteen toteuttamisessa.

Suurin hankaluus menettelytavan hyödyntämisessä esimerkiksi suunnittelua tai arviointia tukevana menettelynä on lähestymistavan akateemisuus: EN-IEC 61508 on todettu filosofiseksi. TET-menettelyllä on muitakin heikkouksia:

- eheystason määrääminen on vaikeaa, koska riskit ovat huonosti yhteismitattavia
- eheystason määrittelyn arviointikriteerit puuttuvat
- eheystason määrittelyssä ei oteta hyötynäkökohtia huomioon
- alhaisiin eheystasoihin ei kiinnitetä riittävästi huomiota.

Menetelmien (ja tekniikoiden) tehokkuus ja soveltamisehdot riskinvähennyksessä selvitetään. Testausten ja analyysien kattavuus järjestelmän ja ohjelmiston verifioinnissa ja validoinnissa sekä koko luotettavuuskuvan saamisessa on ensiarvoisen tärkeää.

Ohjelmiston monimutkaisuuden liitännät valmisohjelmistojen, uudelleenkäytettävien, perättäiskehitettävien ohjelmistojen laatuun, luotettavuuteen ja taloudellisuuteen ovat mitä ilmeisimmät. Yksinkertainen ohjelma on aina helpommin liitettävissä kokonaisuuteen, vaihdettava, muutettava tai ylläpidettävä. Ts. luotettavuuden suunnitteluun kiinnitetään huomiota. Miten on mittaamisen ja osoittamisen laita? Miten mitata ostettavaksi aiottu ohjelmistotuote? Vai tarvitseeko mitata ollenkaan, riittävätkö laadulliset kokemustiedot. Miten osoittaa? Kenelle?

Kaupallisia valmiskomponenttipohjaisia ohjelmistoja tullaan käyttämään kriittisissäkin sovelluksissa edellyttäen, että niistä on saatavilla riittävät luotettavuustiedot. Atomienergiajärjestö IAEA kiinnittää erityistä huomiota ohjelmistopohjaisten järjestelmien ja erityisesti COTSien lisääntyvään merkitykseen ydinvoimalaitosten turvallisuudelle.

Huolimatta ohjelmistotyypistä vaatimusten luotettavuudelle ja muille luotettavuusattribuuteille tulee täyttyä.

Tehokas ohjelmistojen uudelleenkäyttö on kuitenkin ongelmallista. Ohjelmistojen integroitavuus ja luotettavuus eivät ole riittävän korkealla tasolla. Komponenttiohjelmistoilla on mahdollisuus päästä tehokkaaseen uudelleenkäyttöön, jos niitä pystytään hyödyntämään nopeassa sovelluskehityksessä integroitavasti ja luotettavasti sekä lisäksi siten, että ne tukevat ohjelmiston laadunvarmistusta.

Komponenttiohjelmistot muuttavat siten myös luotettavuuden arviointikuvaa. Keskittyminen komponenttiohjelmistoon uudelleenkäytettävänä, mahdollisesti myytävänä pakettina, merkitsee myös keskittymistä osaamiseen, dokumentointiin ja toimintatapoihin. Näitä kaikkia keskittymistarpeita tarvitaan, jotta komponenttiohjelmistosta tulisi integroituva ja luotettava.

Komponenttiohjelmistojen luotettavuuteen liittyvät tarpeet:

- tietämyksen hankkiminen
- arvio siitä, miten strategisesti merkittävää luotettavuus on ohjelmistotuotannossa
- komponenttiohjelmiston luotettavuus muutoksien jälkeen, sillä muutostyöt helposti aiheuttavat uusia virheitä
- luotettavuustiedon uudelleenkäytettävyys
- suorituskyvyn ja luotettavuuden ristiriitaisuuden ratkaiseminen
- integroitavuuden ja luotettavuuden ristiriitaisuuden ratkaiseminen
- koulutustarve käytännön projekteissa.

Luotettavuus liittyy seuraaviin komponenttiohjelmiston tukiteknologioihin:

- ohjeistus: standardointi, liitynnät ja toiminnallinen määrittely, katselmointikäytännöt sekä määrittelyn, (erityisesti) arkkitehtuurin ja suunnittelun luotettavuustekninen päätöksenteko
- menetelmät: luotettavan määrittelyn, arkkitehtuurin, suunnittelun, toteutuksen (tekeminen ja ensisijassa) tarkastaminen, luotettava oliomallinnus ja kielet, luotettava integrointitestaus.

Projektissa keskityttiin luotettavuuden arviointiin ohjelmistointegroijan ja loppukäyttäjän näkökulmasta. Integroijan päämääränä on koota valmiskomponenttipohjainen ohjelmisto, jonka luotettavuus on kyetty arvioimaan. Valmisohjelmistot ovat mustia laati-

koita, joista ei useinkaan ole saatavilla riittävästi verifiointi- ja validointitietoja. Integroijan ongelmia ovat seuraavat:

- Monimutkaisuuden kasvaminen johtuen siitä, että COTSit ovat ulkoisesti hölliä, niillä on helppo toteuttaa uusia toimintoja.
- Nopea vanhenevuus digitaaliteknologian nopean uusiutumisen vuoksi. Tarve vanhojen osien uusimiselle on välttämättömämpää kuin analogisilla laitteilla.
- Muutosten hallinta vaikeaa, mikä johtuu monien saatavilla olevien COTS-tuotteiden patenttioikeuksista. Valmistajat saattavat tehdä sisäisiä tuotemuutoksia ilmoittamatta siitä käyttäjille. He olettavat, että jos tuote on edelleen sopiva ja sillä on oikeat toiminnot ja ominaisuudet, muutoksista ei tarvitse ilmoittaa. Monilla aloilla kuitenkin on tiukat konfiguraatio- ja versiohallinnan vaatimukset ja kyseinen muutostoiminta voi tuottaa ongelmia.
- Dokumentaation puuttuminen johtaa vaikeuksiin, koska monissa tapauksissa tarvitaan tietoa mm. tuotteen laadun ohjaamisesta, ohjelmointistandardeista, ja verifiointimenetelmistä ja -tuloksista. Dokumentteja joko ei ole saatavilla mistään hinnasta tai ne ovat hyvin kalliita.
- Ylimääräiset tai tahattomat toiminnot COTS-tuotteiden monikäyttöisyyden vuoksi. Nämä toiminnot voivat aiheuttaa odottamattomia ja testaamattomia toimintoja. Tahattomat toiminnot jäävät tuotteeseen monimutkaisen suunnittelun pohjalta. Niitä ei huomata ottaa testauksiin mukaan.
- Suurten toimittajien innottomuus myydä tuotteita, joiden menekki on vähäistä, tai jos menekki on suurta, niin he eivät mielellään toimita laatu- ja luotettavuustietoja COTS-tuotteistaan.

Komponenttipohjainen ohjelmistokehitys kattaa tietyn elinkaarimallin, johon kuuluvat perinteisten elinkaarivaiheiden lisäksi sopivien COTSien hakeminen, valitsemisen ja integrointi sekä myös päivittäminen. Päivittämiseen on varauduttava jo kehityksen aikana, sillä päivittämistarve on yksi niistä erityispiirteistä, joissa komponenttipohjainen ohjelmistokehitys selvästi eroaa perinteisestä ohjelmistokehityksestä. Valmisohjelmistojen valmistajien elinehto on tuottaa uusia komponenttiversioita; puhutaankin perättäiskehittämisestä eli evolutionaarisesta kehittämisestä.

Komponenttipohjaisen ohjelmistokehityksen elinkaarimallin vaiheistus vaatisi laadun ja luotettavuuden arviointia, mutta hyvät menetelmät puuttuvat. Perättäiskehittämisen suurimmat ongelmat ovat luotettavuuden heikkeneminen jokaisen julkistetun tuoteversion jälkeen. Heikkeneminen näkyy selvästi lisääntyneenä testaus- ja ylläpitotarpeena, mikä ylittää saavutetut hyödyt suunnittelun ja koodauksen työmäärän vähentymisestä.

Valmisohjelmiston jatkuva muuttaminen edellyttää tiettyjä menetelmiä, tekniikoita ja työkaluja toiminnallisuuden, suorituskyvyn, luotettavuuden ja muiden ei-toiminnallisten ominaisuuksien ylläpitämiseksi ja muuttamiseksi luotettavasti ja kustannustehokkaasti. Miksi kehitystä tarvitaan? Mitkä ovat kehittämisprosessin pääominaisuudet? Mitkä ovat kehittämisen vaikutukset ohjelmistoprosessille ja -tuotteille? Mitkä ovat vaikutukset prosessien ja kehittämisen jatkuvaan hallintaan?

Kustannusmalleissa tulisi myös ohjelmiston muutostiheys ottaa huomioon. Perättäiskehittämistä on osattava hallita, sillä jatkuva monimutkaistuminen ja vanheneminen voivat olla merkittäviä luotettavuus- ja kustannustekijöitä. Palautevaateet, johdon politiikka, organisatoriset projekti- ja tiimirakenteet esittävät merkittäviä kustannuslähteitä, joiden kontribuutiot vaihtelevat järjestelmittäin ja sovelluksittain. Perättäiskehittäminen on hyvin dynaaminen prosessi, joka vaihtelee ajan mukana niin, että staattisin mallein voi olla vaikea saada niistä otetta.

Ohjelmiston laatutakuita voivat antaa myös kolmannet osapuolet. CERD-tutkimusprojektissa selvitettiin, mitä arviointi- ja sertifiointilaitos lupaavat, jos heidän takuunsa pettävät. Asia on monimutkainen siinäkin mielessä, että sertifiointilaitos tai yleensä takuun antaja haluaa välttyä sanktioiden lisäksi myös aikaa vieviltä käsittelyprosesseilta mm. oikeusistuimissa. Käsitteiden täsmällisen määrittämisen ja luotettavuuden mittaamisen tulisi olla keskeinen takuuprosessin välikappale.

Testien kattavuus on yksi merkittävimpiä luotettavuustekijöitä. Myös käytännöllisimmät ohjelmiston luotettavuuden arviointimenetelmät perustuvat testauksiin. Niistä on osoituksena mm. viitteiden Musa et al. (1998) ja Voas & Payne (2000) esittämät luotettavuusmallit. Riittävätkö testitulokset ja -arvioinnit takuun pohjaksi? Useissa lähdeteoksissa esitetään näkemyksiä ettei ohjelmaa kyetä täysin testaamaan kaikkien virheiden löytämiseksi.

Testausten kattavuus yhdessä muiden verifiointimenetelmien kanssa onkin ratkaisevassa asemassa ohjelmiston luotettavuuden arvioinnissa. Riskienhallintaan ja riskianalyyseihin kiinnitetään joissakin yrityksissä erityistä huomiota, mutta useimmissa suhtautuminen niihin on hyvin ponnetonta. Käytännöllisimmät menetelmät perustuvat testauksiin, joista on hyvänä esimerkkinä Musan (1998) ja Voas & Payne'n (2000) esittämät mallit. Kuitenkin useissa lähdeteoksissa esitetään samankaltaisia käsityksiä siitä, ettei ohjelmaa kyetä täysin testaamaan kaikkien virheiden löytämiseksi. Testaamisella ei myöskään kyetä osoittamaan virheiden poissaoloa, mihin taas riskienhallintaan ja -analyyseihin perustuvat mallit nimenomaan tähtäävät (Harju 2000). Takuurajojen nostamista varten vakuuttautuminen riittävästä luotettavuudesta on ensiarvoisen tärkeä, ja siten tärkeysasteesta ja resurssimahdollisuuksista riippuen tulisi käyttää sekä testaamiseen että riskianalyyseihin pohjautuvia luotettavuuden arviointimalleja.

Lähdeluettelo

Ammann, P. & Knight, J. C. 1998. Data diversity: an approach to software fault tolerance. IEEE Transaction on Computers. s. 418–425.

ANSI/ISA-S84.01, Standard. 1996. Application of safety instrumented systems to the process industries. American Society of Mechanical Engineers, ANSI, Instrument Society of America, ISA.

ATU, Suomen Automaation Tuki Oy. 2001. Laatu automaatiossa. Parhaat käytännöt. Tommila, T. (ed.). Helsinki: Suomen Automaatioseura ry. 245 s. ISBN 952-5183-12-2.

Ayala, J. 1998. Training the Microsoft Way. Windows NT Magazine. s. 122–129. (March 1998.)

Basili, V., Briand, L. & Melo, W. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering. Vol. 22, No. 10, s. 751–761.

Basili, V. R. & Perricone, B. T. 1984. Software Errors and Complexity: An Empirical Investigation. Communication of the ACM. Vol. 27, No. 1, s. 42–52.

Beizer, B. 1990. Software Testing Techniques. Van Nostrand Reinhold. 503 s. ISBN 0-442-20672-0.

Benso, A., Rebaudengo, M. & Reorda, M. S. 1999. FlexFi: A Flexible Fault Injection Environment for Microprosessor-Based Systems. SAFECOMP'99. Berliin: Springer-Verlag. S. 323–335. (LNCS 1698.) ISBN 0302-9743.

Blum, M. & Kannan, S. 1995. Designing programs that check their work. JACM. S. 269–291.

Boehm, B. W. 1981. Software Engineering Economics. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. W. 1988. A spiral model of software development and enhancement. Computer. Vol. 21, No. 5, s. 61–72.

Boehm, B. W. 1999. Making RAD working for your project. Computer. Vol. 22, No. 3, s. 113–114.

Briand, L., Devanbu, P. & Melo, W. 1997. An investigation into coupling measures for C++. Proceedings of the 19th International Conference on Software Engineering.

Briand, L., Wuest, J., Daly, J. & Porter, V. 2000. Exploring the relationships between design measures and software quality in object oriented systems. Journal of Systems and Software. Vol. 51, s. 245–273.

CFR, Code of Federal Regulations. 1975. Quality Assurance Criteria for Nuclear Power Plants and Fuel Processing Plants. National Archives and Records Administration, Title 10 of the Code of Federal Regulations, Part 50, Appendix B, Office of the Federal Register.

CFR, Code of Federal Regulations. 1995. Reporting of Defects and Noncompliance. National Archives and Records Administration, Title 10 of the Code of Federal Regulations, Part 21, Office of the Federal Register.

Chidamber, S. & Kemerer, C. 1994. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering. Vol. 20, No. 6, s. 476–493.

Clark, J. & Pradhan, D. 1995. Fault Injection: A method for validating Computer-System Dependability. IEEE Computer. June 1995. s. 47–56.

DO-178B, Federal Aviation Authority. 1992. Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronatutics, RTCA, Inc. (RTCA/DO-178B.)

Dustin, E., Rashka, J. & Paul, J. 2000. Automated Software Testing. 3. p. Toronto: Addison-Wesley. 549 s. ISBN 0-201-43287-0.

Emam, K. E., Benlarbi, S., Goel, N. & Rai, S. N. 2001a. Comparing case-based reasoning classifiers for predicting high risk software components. The Journal of Systems and Software. Vol. 55, s. 301–320.

Emam, K. E., Melo, W. & Machado, J. C. 2001b. The prediction of faulty classes using object-oriented design metrics. The Journal of Systems and Software. Vol. 56, s. 63–75.

EN-IEC 61508, standard. 2001. Functional safety of programmable electronic systems: International Electrotechnical Commission. Parts: 1–7.

EPRI, Electric Power Research Institute. 1988. Guideline for the Utilization of Commercial Grade Items in Nuclear Safety Related Applications (NCIG-07), EPRI NP-5652.

EPRI, Electric Power Research Institute. 1993. Guidelines on Licensing Digital Upgrades, EPRI TR-102348.

EPRI, Electric Power Research Institute. 1994. Supplemental Guidance for the Application of EPRI NP-5652 on the Utilization of Commercial Grade Items, EPRI TR-102260.

FDA, Center for Devices and Radiological Health. 1998. Guidance for Off-the-Shelf Software Use in Medical Device. U.S Department of health and human services. Food and Drug Administration. Draft Guidance.

Fenton, N.E. 1995. Software Measurement: A necessary scientific basis. Predictably Dependable Computing Systems. Berlin: Springer-Verlag. S. 67–86. ISBN 3-540-59334-9.

Fenton, N. E. & Neil, M. 1999. Software Metrics: successes, failures and new directions. The Journal of Systems and Software. Vol. 47, s. 149–157.

Fioravanti, F. & Nesi, P. 2000. A method and tool for assessing object-oriented projects and metrics management. The Journal of Systems and Software. Vol. 53.

Halstead, M. H. 1977. Elements of Software Science. New York: Elsevier-North Holland.

Hamlet, D., Mason, D. & Woit, D. 2001. Theory of Software Reliability Based on Components. Proceedings 23rd International Conference on Software Engineering ICSE'2001. 10 s.

Harju, H. 2000. Ohjelmiston luotettavuuden kvalitatiivinen arviointi. Espoo: Valtion teknillinen tutkimuskeskus. VTT Tiedotteita 2066, 111 s. ISBN 951-38-5766-2. (http://www.inf.vtt.fi/pdf).

IAEA, Technical report. 1999. Use of commercial off-the-shelf (COTS) digital items in the upgrading/modernization of I&C systems important to safety including regulatory requirements and safety assessment. International Atomic Energy Agency. 34 s. (Draft IAEA-TECDOC-XXX.)

IAEA, Technical report. 2001. Scientific basis and engineering solutions for cost-effective assessments of software based I&C systems. (IAEA-CRP-I2.10.11-2.)

IEEE 982.2, standard. 1988. Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software.

IEEE, standard. 1993. Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations, IEEE 7-4.3.2-1993, Institute of Electrical and Electronics Engineers, Piscataway, NJ.

Jaaksi, A., Aalto, J.-M., Aalto, A. & Vättö, K. 1999. Object Development. Industry-Proven Approaches with UML. Tried & True. Cambridge, UK: Cambridge University Press. 315 s. ISBN 0-521-64530-1.

Jones, C. 1996. Programming Languages Table. Software Productivity Research, Inc. Saatavilla maaliskuussa 2001: http://www.spr.com/library/0langtbl.htm (Release 8.2, March 1996).

Kaner, C. & Pels, D. 1998. Bad software. What to do when software fails. New York: Wiley. 365 s. ISBN 0-471-31826-4.

Knight, J., Class, A., Fernandez, A. & Wika, K. 1994. Testing a safety-critical applications. Proceedings ISSTA'94. Seattle, WA.

Krishnamurthy, L. & Mathur, A. 1997. The estimation of system reliability using reliabilities of its components and their interfaces. Proceedings 8th International Symposium on Software Reliability Engineering. Albuquerque, NM, USA.

Kuikka, S. 1999. A batch process management framework. Domain-specific, design pattern and software component based approach. Espoo: Technical research centre of Finland. VTT Publications 398, 215 s.

Laprie, J.-C. 1984. Dependability evaluation of software system in operation. IEEE Transaction on Software Engineering. Vol. 10, No. 6, s. 701–714.

Laprie, J.-C. 1998. Dependability: Basic Concepts and Terminology. Dependability Handbook. [Toulouse]: Laboratory for Dependability Engineering. 290 s. (LAAS Report no 98-346.)

Laprie, J.-C. 1999. Diversity for Dependability. Invited Talk. SAFECOMP'99. Toulouse, France, 29.9.1999.

Lindqvist, U. & Jonsson, E. 1998. A Map of Security Risks Associated with Using COTS. IEEE Computer. s. 60–66. (June 1998.)

Lindsay, P. A. & McDermid, J. A. 1997. A systematic approach to software safety integrity levels. In: Daniel, P. (ed.). Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP'97), Berlin: Springer. S. 70–82.

Littlewood, B. 1979. Software reliability model for modular program structure. IEEE Transaction on Reliability Engineering. Vol. 28, No. 3. s. 241–246.

Lorenz, M. & Kidd, J. 1994. Object-Oriented Software Metrics. Englewood Cliffs, NJ.: Prentice-Hall. 146 s.

Lyu, M. (ed.). 1996. Handbook of Software Reliability Engineering. McGraw-Hill, New York.

Mason, D. V. & Woit. M. 2000. Using Partition Analysis to Legitimize Software Reliability Measurement. Saatavilla internetistä: http://www.sarg.ryerson.ca/sarg/papers/.

McCabe, T. J. 1976. A Complexity Measure. IEEE Transaction on Software Engineering. Vol. SE 2, No. 4, s. 308–320.

Mills, H. 1988. Software Productivity. Dorset House Publishing, New York, S. 13–18.

MIL-STD-882C, US Department of Defence. 1993. System Safety Program Requirements.

MoD 00-55, Defence Standards. 1997. Requirements for Safety Related Software in Defence Equipment. U.K. Ministry of Defence.

MoD 00-56, Defence Standards. 1996. Safety Management Requirements for Defence Systems. U.K. Ministry of Defence.

Moller, K-H. & Paulish, D. 1993. An empirical investigation of software fault distribution. Proceedings of the First International Software Metrics Symposium. S. 72–80.

Munson, J. C., Khoshgoftaar, T. M. 1991. The Use of Software Complexity Metrics in Software Reliability Modeling. Proceedings of 2nd International Symposium on Software Reliability Engineering. Austin, TX, USA: IEEE Computer Society Press. S. 2–11. (ISSRE'91.)

Musa, J. D., Iannino, A. & Okumoto, K. 1987. Software Reliability. Measurement, Prediction, Application. McGraw-Hill, New York. 621 s.

Musa, J. 1998. Software Reliability Engineering. New York: McGraw-Hill. 391 s.

NASA, Guidebook. 1989. Software Assurance Guidebook, NASA GSFC MD, Office of Safety and Mission Assurance.

Niemelä, E., Kuikka, S., Vilkuna, K., Lampola, M., Ahonen, J., Forssel, M., Korhonen, R., Seppänen, V. & Ventä, O. 2000. Teolliset komponenttiohjelmistot. Kehittämistarpeet ja toimenpide-ehdotukset. TEKES. Teknologiakatsaus 89/2000. ISSN 1239-758X. 129 s.

Ohlsson, N. & Alberg, H. 1996. Predicting fault-prone software modules in telephone switches. IEEE Transactions on Software Engineering. Vol. 22, No. 12, s. 886–894.

Ottenstein, L. J. & Fodsick, L. D. 1979. Quantitative Estimates of Debugging Requirements. IEEE Transaction on Software Engineering. Vol. SE 5, No. 5. s. 504–514.

Palady, P. 1998. Failure Modes & Effects Analysis, Author's Edition. USA: Practical Applications. 262 s. ISBN 0-9663160-0-2.

Paulk, et al. 1993. Capability Maturity Model, Vesion 1.1. In: IEEE Software. July 1993. S. 18–27.

Pöyhönen, I., Harju, H., Kemppainen-Kajola, P., Kuhakoski, K., Kylmälä, K., Spankie, G. & Ventä, O. 2002. Vaatimukset ohjelmistoa sisältäville lääkintälaitteille. Hallinta ja menetelmät vaatimustenmukaisuuden osoittamiseksi. (Ilmestyy VTT Tiedotteissa vuoden 2002 aikana.).

Prechelt, L. 2000. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl. Submission to IEEE Computer. 7 s. (14.4.2000 .)

EN 50126, Standard. 1999. Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS).

EN 50128, Standard. 1999. Railway applications – Communications, signalling and processing systems - Software for railway control and protection systems.

Pressman, R. S. 1997. Software Engineering: A Practioner's Approach. 4. p. McGraw-Hill. 885 s. IBSN 0-07-114603-2.

SFS-EN-60601-1-4, standard. 1996. Medical Electrical Equipment--Part 4: Collateral Standard: Programmable Electrical Medical Systems. 64s.

SFS-EN ISO 9000-3, standardi. 1997. Laatujohtamisen ja laadunvarmistuksen standardit. Osa 3: Suuntaviivat standardin SFS-EN ISO 9001 soveltamiseksi ohjelmistojen kehittämisessä, toimittamisessa ja ylläpidossa. Helsinki: Suomen Standardisoimisliitto, SFS. 34 s.

SFS-EN ISO 9001, standardi. 1994. Laatujärjestelmät. Suunnittelun, tuotekehityksen, tuotannon, asennuksen ja huollon laadunvarmistusmalli. Helsinki: Suomen Standardisoimisliitto, SFS. 31 s.

Takahashi, M. & Kamayachi, Y. 1989. An Empirical Study of A Model for Program Error Prediction. IEEE Transaction on Software Engineering. Vol. SE-15, No. 1, s. 82–89.

Talbert, N. 1998. The Cost of COTS, Interview with John McDermid . Computer, Vol. 31, No. 6, S. 46–52.

UL, standard. 1998. Standard for Software in Programmable Components. Underwriter's Laboratory Inc. May 29, 1998.

Vigder, M. R. 1998. Inspecting COTS Based Software Systems: Verifying an Architecture to Support Management of Long-Lived Systems. National Research Council Canada, Institute for Institut Information Technology. NRC Report No. 41604. 20 s.

Vigder, M. R., Dean, J.C. 1997. An Architectural Approach to Building Systems from COTS Components. Proceedings of the 22nd Annual Software Engineering Workshop. Greenbelt, Maryland: National Aeronautics and Space Administration - Goddard Space Flight Center. 14 s.

Vigder, M. R., Dean, J.C. 1998. Building Maintainable COTS Based Systems. National Research Council Canada, Institute for Institut Information Technology. NRC Report No. 41611. 8 s.

Voas, J. 1998a. The Challenges of Using COTS Software in Component-Based Development. Computer. Vol. 31, No. 6, S. 44–45.

Voas, J. 1998b. Software Certification Laboratories? Saatavilla kesäkuussa 2001: http://www.cigital.com/research/jmv.html. 4 s.

Voas, J. & Miller, K. 1995. Software testability: The new verification. IEEE Software. Vol. 12, No. 3, s. 17–28.

Voas, J. & Payne, J. 2000. Dependability Certification of Software Components. Teoksessa: The Journal of Systems and Software. Vol. 52, s. 165–172.

Wallace, D. R. & Ippolito, L. M. 1994. A framework for the development and assurance of high integrity software, NIST SP 500-223, National Institute of Standards and Technology, Gathersburg, MD 20899. Saatavilla kesäkuussa 2001: http://hissa.nist.gov/publications/sp223/

Wallace, D. R., Ippolito, L. M. & Cuthill, B. 1996. Reference information for the software verification and validation process, NIST SP 500-234, National Institute of Standards and Technology, Gathersburg, MD 20899. Saatavilla kesäkuussa 2001: http://hissa.nist.gov/VV234/

Wallace, D. R. & Kuhn, D. R. 2000. Lessons from 342 medical device failures. Saatavilla kesäkuussa 2001: http://hissa.nist.gov/effProject/

Wallnau, K. C., Carney, D. & Pollak, B. 1998. COTS Software Evaluation. Software Engineering Institute. Saatavilla: http://www.idi.ntnu.no/emner/sif80at/curriculum/ICCBSS-sumit.pdf

Woit, D. & Mason, D. 1999. Setting software reliability back 20 years. Ryerson Polytechnic University. Saatavilla kesäkuussa 2001: http://www.sarg.ryerson.ca/sarg/papers/199906-csr/

Zeigler, S. 1995. Comparing Development Costs of C and Ada. Rational Software Corporation, March 30, 1995, Saatavilla kesäkuussa 2001: http://www.adaic.org/docs/reports/cada/cada_art.html

Zhang, X. & Pham, H. 2000. An analysis of factors affecting software reliability. The Journal of Systems and Software. Vol. 50, s. 43–56.

Zuse, H. 1990. Software complexity: Measures and Methods. McGruyter, New York.

Liite A: Taulukot

Taulukko A1. Ohjelmiston luotettavuuteen vaikuttavat tekijät tärkeysjärjestyksessä tutkimuksen (Zhang & Pham 2000) mukaan.

Rank	Factor name	Normalized priorities
1	Program complexity	0.0376
2	Programmer skills	0.0369
3	Testing coverage	0.0367
4	Testing effort	0.0365
5	Testing environment	0.0353
6	Frequency of spec change	0.0348
7	Testing methodologies	0.0343
8	Requirements analysis	0.0341
9	Percentage of reused code	0.0336
10	Relationship of detailed design and requirement	0.0332
11	Level of programming technologies	0.0331
12	Documentation	0.0328
13	Program work load	0.0327
14	Testing tools	0.0322
15	Programmer organization	0.0321
16	Domain knowledge	0.0317
17	Difficulty of programming	0.0317
18	Design methodologies	0.0317
19	Human nature (mistake and omission)	0.0316
20	Development management	0.0316
21	Testing resource allocation	0.0309
22	Amount of programming effort	0.0307
23	Program categories	0.0305
24	Work standards	0.0298
25	System software	0.0283
26	Volume of program design Documents	0.0275
27	Development team size	0.0273
28	Programming language	0.0271
29	Processor	0.0241
30	Telecommunication device	0.0240
31	Input/output device	0.0229
32	Storage device	0.0212

Taulukko A2 .Ohjelmiston luotettavuustekijöiden korrelaatiot (Zhang & Pham 2000).

Name of factors	Correlated factors
Program complexity	Development team size
Programmer skills	Difficulty of programming; Program work load
Testing coverage	Program categories
	Testing methodologies
Testing effort	Level of programming technologies
	Work standards; Testing environment
	Development management
Testing environment	Level of programming technologies
	Relationship of detailed design and requirement
	Work standards; Testing effort
Testing methodologies	Testing coverage; Testing tools
Requirements analysis	Relationship of detailed design and requirement
	System software
Percentage of reused code	Programming language; Program work load
	Volume of program design documents
	Development management
	Testing resource allocation
	Testing methodologies; Testing tools
	Documentation
Relationship of detailed design and requirement	Requirements analysis
	Testing environment
Level of programming technologies	Testing environment; Testing effort
Documentation	Percentage of reused code
	Testing tools
	Telecommunication device
Program work load	Difficulty of programming; Programmer skills
	Amount of programming effort
	Programming language; Work standards
	Testing resource allocation; Testing tools
	Input/output device
Testing tools	Percentage of reused code
	Program work load
	Documentation
Programmer organization	
Domain knowledge	

Taulukko jatkuu

Name of factors	Correlated factors
Difficulty of programming	Percentage of reused code
	Programmer skills; Program work load
Design methodologies	Program categories
Human nature (mistake and omission)	
Development management	Testing effort
Testing resource allocation	Percentage of reused code
	Testing methodologies
Amount of programming effort	Percentage of reused code
	Programming language; Program work load
Program categories	Design methodologies
	Testing coverage
Work standards	Development management
	Program work load
	Testing environment; Testing effort
System software	Requirements analysis
	Processor
	Input/output device
	Telecommunication device
Volume of program design documents	Percentage of reused code
Development team size	Program complexity; Program work load
Programming language	Amount of programming eflon
	Percentage of reused code
	Program work load
Processor	Storage device
	Telecommunication device
Telecommunication device	Documentation
	Processor
	System software
Input/output device	Processor
Storage device	Input/output device

Taulukko A3. Esimerkkejä lääkintälaitteiden vikojen ennaltaehkäisevistä ja havainnoivista menetelmäsuosituksista.

Vikaluokka	Vikatyyppi	Ehkäiseminen	Havaitseminen
Logiikka	Ohjauksen logiikkavirheet tai -puutteet	Jäljitettävyysanalyysi, suunnittelun ja toteutuksen vertaaminen	Koodin katselmointi, tarkistaminen, testit
Laskenta	Väärinkoodatut vakiot	Koodin lukeminen	Lukeminen, yksikkötestit
Muutoksen vaikutus	Puuttuva suunnittelun ja toteutuksen välinen verifiointi	Jäljitettävyysanalyysi, muutosten hallinta	Muutosten tarkistaminen, regressiotestit
Data	Virheellinen input-data	Arvoalueiden määrittely, toimenpiteet poikkeustapauksissa	Input-määrittelyn verifiointi, testit
Virheellinen tai puuttuva vaatimus	Vaatimusmäärittely ei kattanut erikoistilanteita	Mallinnus, formaalit menetelmät, jäljitettävyys	Vaatimuskatselmukset, järjestelmätestit
Toiminnon puuttuminen	Osa järjestelmäfunktioista puuttuu	Määrittely, jäljitettävyysanalyysi	Jäljitettävyysanalyysi, testit
Muu virhe	Kirjoitusvirhe koodissa aiheuttaa laiteyhteensopimattomuuden	Koodin lukeminen	Algoritmien läpikäynti, testaaminen
Laadunvarmistus	Testaussuunnitelmaa ei noudatettu	Projektin johtamisen tarkistaminen	Projektin tilanteen seuranta
Ajastus	Epäsynkronisuus	Simulointi, suunnittelun katselmointi	Ajastuksen analysointi, testit
Alustus	Alustusarvojen talletttaminen epäonnistuu	Käyttöönoton ja jatkuvan käytön ehtojen määrittely	Koodin lukeminen, kuormitustestit
Liityntävirhe	Ohjelmiston laiteliittymä tai yhteydet muihin ohjelmistoihin eivät toimi	Vaatimusmäärittely, interface-ehtojen jäljittäminen	Katselmukset, järjestelmätestit
Rakenteenhallinta	Väärä lähdekoodi	CM-välineiden käyttö	CM-vastaava tarkistaa version
Vikasietoisuus	Ylikuormitus kaataa ohjelman tai laitteen	Vikasietoisuuden rakentaminen, kuormitusrajojen määrittely	Kuormitustestit, epänormaalien inputarvojen syöttäminen

Taulukko A4. Otteita luotettavuusattribuuteille kohdistuvista viitteen (Vigder 1998) tarkistuslistoista. Viite sisältää tietoa kategorioiden määrittelystä, tavoitteista, sekä tarkistuskohteiden kuvauksista.

Category, *Definition,* Objective	**Checklist**
Connector infrastructure *Medium by which information and control is transferred between components.* The objectives in inspecting the connector architecture: – determine whether components can be easily moved to allow for reconfiguration of the system; – verify that components can be easily added, removed, and substituted; – maximize the selection of COTS vendors who can provide components that are compatible with the system; – determine the level of visibility the connector provides for the purposes of troubleshooting and testing.	Have all connector types been identified and documented? Have the number of connector types been minimized? Has the security of the connectors been evaluated? Can new security mechanisms be integrated into the connector? Is it possible to monitor behaviour on the connectors? Can instrumentation be added to connectors? Are the connector types supported by different vendors? Have the connectors been evaluated as to the support in the vendor community? How are the components bound to the connectors and to each other? Are the connections synchronous or asynchronous?
Interconnection topology *A map of the way the components communicate with each other.* The objective of evaluating the interconnection topology is to minimize the number of interconnections and to simplify the interconnection patterns.	Are the interconnections identified and documented? Can interconnection topology be simplified? The more complex the interconnection topology of a system the more difficult to integrate, test, and troubleshoot.
Interfaces *Interfaces define the format of the information that is passed between components through the connectors.* The objective of the architecture is to verify that interfaces provide: – Interfaces remain independent of the underlying component so that component substitution is possible. – Interfaces facilitate the instrumentation, monitoring, testing, and troubleshooting of systems. – Interfaces are documented, versioned, and under configuration management.	Are standard interfaces available for the component? Are the interfaces separate from the implementation? Are all accesses to the component through the accepted interface? Are the interfaces under the control of the integrator/user? Are adapters (wrappers) being used? Are the interfaces versioned? Is run-time compatibility checking performed? Is behaviour at the interfaces observable? Does the component have an instrumentation interface?

Taulukko jatkuu

Category, *Definition,* Objective	**Checklist**
Instrumentation *Instrumentation is the ability to observe and monitor the behaviour of a system during execution.* The purpose of evaluating the instrumentation is the following: – Determine whether integrators and users have an adequate instrumentation capability; – Determine what level of instrumentation is available from the architecture; – Determine the usefulness of standards for instrumentation.	When evaluating components, determine the level of instrumentation that has been included in the component? Can faults be injected for testing/ evaluation/ troubleshooting? For testing and troubleshooting it should be possible to inject faults into a system. Has instrumentation been added to the glue code? How is instrumentation controlled by the system manager?
Configuration management *Configuration management of a COTS based system is managing the deployment of component versions at each site.* CM in a COTS based system differs from configuration management for a developer or maintainer. CM for COTS based systems is not a case of managing versions and variants of the source code, but of managing at a level where component versions are the principal means of tracking. The goal of the configuration management system is to track the status of each installed system.	What is the mechanism by which the different versions of each COTS component are tracked? Is there a method for recording the known incompatibilities and compatibilities between specific versions of COTS components? Is there a process for recording which versions of software components work correctly together, and which do not? Is there run-time checking by which components verify the correct versions of the other components with which they communicate? Is run-time version checking included? Is there a means by which a system manager can determine the versions of components that are deployed at each site?

Liitteen A lähdeluettelo

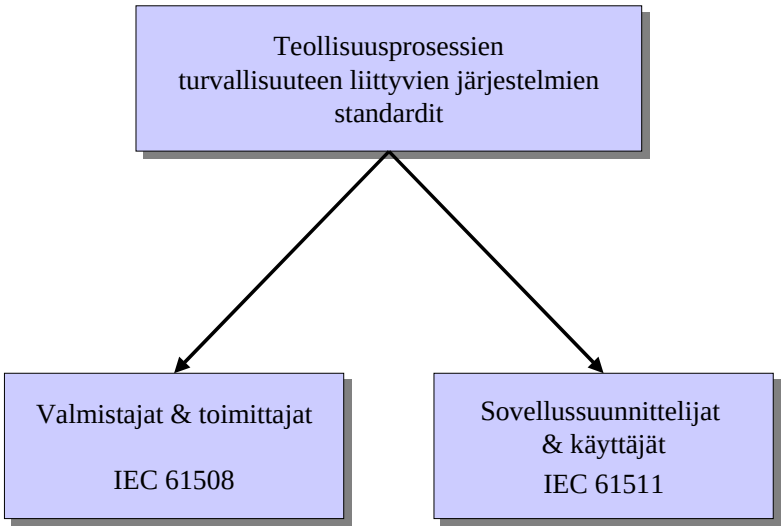
Vigder, M. R. 1998. Inspecting COTS Based Software Systems: Verifying an Architecture to Support Management of Long-Lived Systems. National Research Council Canada, Institute for Institut Information Technology. NRC 41604. 20 s.

Zhang, X. & Pham, H. 2000. An analysis of factors affecting software reliability. The Journal of Systems and Software. Vol. 50, s. 43–56.

Liite B: IEC 61508:n turvallisuuden eheystasomenettely

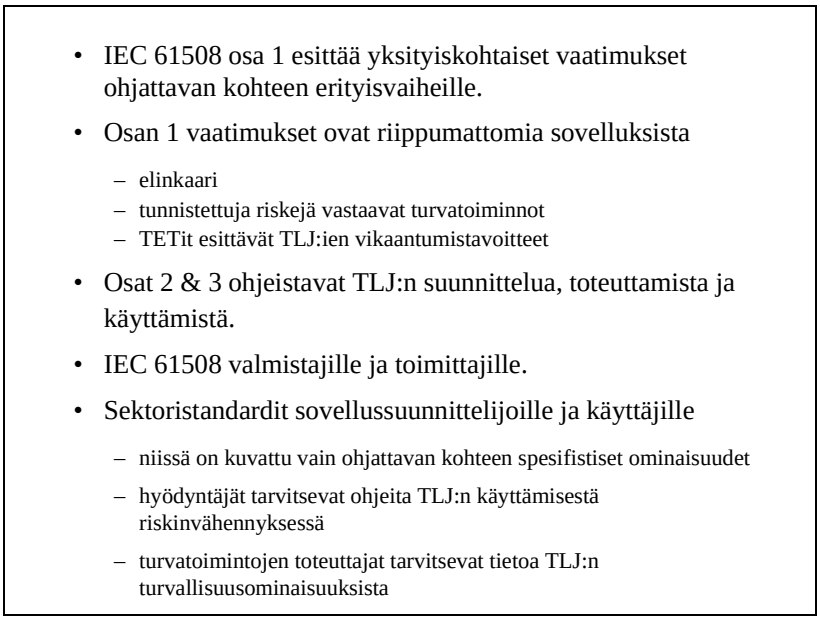
Tässä liitteessä esitetään standardin EN-IEC 61508:n turvallisuuden eheystasomenettelyn pääpiirteet. Standardi on automaatioalalla hyvin laajasti käytetty turvallisuusohje. Mielipiteet sen hyvyydestä vaihtelevat. Jotkut ovat sitä mieltä, että se on liian monimutkainen ja laaja, toisten mielestä siitä puuttuvat monet merkittävät yksityiskohtaiset tiedot, joita tarvitaan toiminnallisen turvallisuuden saavuttamiseksi ja osoittamiseksi.

EN-IEC 61508 on yleisstandardi, jonka viitoittamalle tielle on tarkoitus laatia sektorikohtaisia alastandardeja (kuva B1), joissa yksityiskohdat ja sektorin ominaisuudet pääsevät paremmin esille kuin yleisstandardissa. Tällä hetkellä on vain muutamia sektoristandardeja valmiina, joten EN-IEC 61508 tehtävänä on palvella sitä käyttäjäkuntaa jolle sektorikohtaista standardia ei ole saatavilla. Tulevaisuudessa, kun sektoristandardit yleistyvät, EN-IEC 61508 saatetaan yleiseen yksityiskohdat välttävään muotoon.



Kuva B1. Ohjeitten vaatiman tiedon laajuutta jaetaan yleisstandardeihin ja sektoristandardeihin.

Yleisstandardin ohjeistamat tiedot on kohdennettu erityisesti automaatiojärjestelmien valmistajille, toimittajille sekä valitsijoille (kuva B2). Siinä ei ole sovelluskohtaisia yksityiskohtia, jotka ovat keskeisiä esimerkiksi prosessiteollisuuden automaation turvallisuusstandardissa IEC 61511.



Kuva B2. Standardin EN-IEC 61508 lähtökohtana on ohjattava kohteen vaatimukset. Ohjattavaan kohteeseen kuuluu myös käyttöautomaatio.

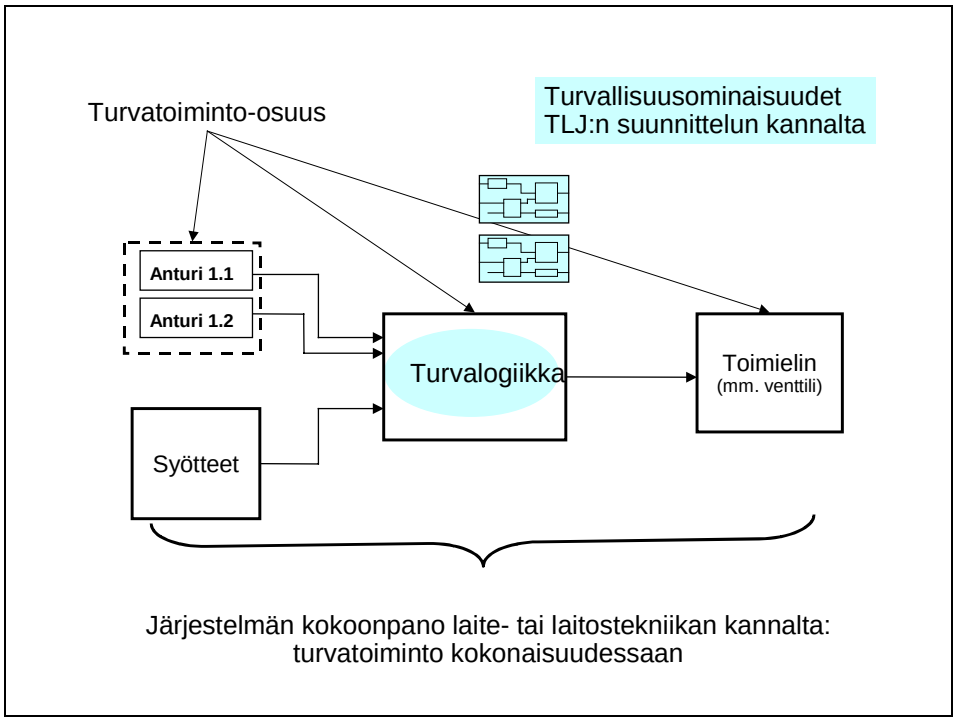
B1. Käsitteet

Turvallisuuteen liittyvä järjestelmä on nimetty järjestelmä, jonka tehtävänä

1) on estää määrättyjen vaaratilanteiden kehittyminen onnettomuudeksi ohjaamalla kohde hallitusti turvalliseen tilaan, tai mikäli turvallista tilaa ei ole, ylläpitää hallitusti vallitsevaa tilaa

2) saavuttaa joko yksinään tai yhdessä muiden TLJ:ien kanssa välttämätön turvallisuuden eheystaso vaadittavien turvatoimintojen toteuttamisessa.

Turvallisuuteen liittyvä järjestelmä sisältää kaiken laitteiston, ohjelmiston ja tukilaitteet, joita tarvitaan määritettyjen turvatoimintojen toteuttamiseksi. Anturit, muut tulolaitteet, toimilaitteet ja muut lähtölaitteet sisältyvät näin ollen turvallisuuteen liittyvään järjestelmään (kuva B3).



Kuva B3. Turvallisuuteen liittyvä järjestelmä koostuu turvatoimintojen kokonaisuudesta. Siihen kuuluu koko ketju antureista toimilaitteisiin, ne mukaan lukien. Turvatoimintojen jakamisessa on mm. kyse vaatimusten jakamisesta turvalogiikalle.

Turvallisuuteen liittyvä järjestelmä voi pohjautua laajaan teknologioiden alueeseen sisältäen sähkötekniikkaa, elektroniikkaa, ohjelmoitavaa elektroniikkaa, hydrauliikkaa ja pneumatiikkaa.[1]

Turvallisuuden eheys on turvallisuuteen liittyvän järjestelmän todennäköisyys suoriutua sille asetetuista turvatoiminnoista määräoloissa vaadittuna aikana. Turvallisuuden eheys luokitellaan neljäksi diskreetisksi tasoksi. Mitä korkeampi turvallisuuden eheystaso on, sitä todennäköisemmin järjestelmä suoriutuu vaadittavista turvatoiminnoista.

Turvallisuuden eheyttä määrättäessä otetaan huomioon kaikki vaaralliseen tilanteeseen johtavien vikaantumisten syyt, esim. laitteistoviat, ohjelmiston aiheuttamat viat ja sähkömagneettiset häiriöt. Jotkut näistä vioista ovat kvantifioitavissa (esim. laitteistovikojen vikataajuudet). Kuitenkin turvallisuuden eheys riippuu myös monista sellaisista tekijöistä, joita ei voida tarkasti kvantifioida. Näitä tekijöitä on tarkasteltava kvalitatiivisesti.

[1] Tässä käsitellään vain sähköteknisiä, elektronisia ja ohjelmoitavia järjestelmiä, joille annetaan yksinkertaistamiseksi lyhenne TLJ.

Turvallisuuden eheys jaetaan kahteen osaan:

1) Turvallisuuteen liittyvän laitteiston eheys, joka voidaan arvioida riittävän tarkasti ja siten laitteiston eheysvaatimus voidaan jakaa osajärjestelmille todennäköisyyspohjaisin normaalisäännöin. Usein laitteiston eheysvaatimuksen saavuttamiseksi on välttämätöntä käyttää redundanssia.

2) Systemaattinen eheys, joka liittyy vaarallisten vikaantumisten systemaattisiin vikoihin ja virheisiin. Systemaattista eheyttä ei yleensä voida riittävän tarkasti kvantifioida. Kvantifioimisvaikeuksien takia sen jakaminen osajärjestelmille ei ole mahdollista. Esimerkiksi satunnaisvikoja vastaan hyvin tehokkaat identtisiin laitteistoihin pohjautuvat redundanttiset menetelmät eivät aina sovi systemaattisia vikoja vastaan. Systemaattinen eheys saavutetaan valitsemalla sopivia menetelmiä ja tekniikoita koko elinkaaren aikana.

Taulukoissa B1 ja B2 on annettu vikaantumismittojen tavoitearvot turvallisuuden eheystasoille.

Taulukko B1. Turvallisuuden eheystasot: turvatoimintojen tavoitearvot harvojen vaateiden toiminnalle.

Turvallisuuden eheystaso	**Harvojen vaateiden toiminnat** Keskimääräinen vikaantumisen todennäköisyys vaadetta kohti
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$

Taulukko B2. Turvallisuuden eheystasot: turvatoimintojen tavoitearvot jatkuvuutta vaativalle toiminnalle.

Turvallisuuden eheystaso	**Tiheiden vaateiden toiminnot** Vaarallisen vikaantumisen todennäköisyys tuntia kohti
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

B2. Turvallisuuteen liittyvän järjestelmän nimeäminen

Lähtökohtana ohjaus- ja suojausjärjestelmän nimeämisessä turvallisuuteen liittyväksi järjestelmäksi ovat turvatoiminnoille määrätyt turvallisuuden eheystasot. Standardi EN-IEC 61508 ohjeistaa turvallisuuteen liittyvän järjestelmän eheystasoksi siihen määrättyjen turvatoimintojen korkeimman turvallisuuden eheystason. Lisäksi turvatoimintoja saa olla vain kohtuullinen määrä, muuten turvallisuuden eheystasovaatimus kasvaa.

Nimeäminen alkaa aina ohjattavan kohteen asettamista vaatimuksista. Ilman TLJ:iä vaaranalaisessa ohjattavassa kohteessa on tietty riski, joka yleensä on suurempi kuin hyväksyttävissä oleva riski. Jotta kohde olisi turvallinen, on riskiä vähennettävä erilaisin toimenpitein vähintään hyväksyttävään riskitasoon saakka. Vaadittava riskin vähentäminen saadaan ohjattavan kohteen sisältämän riskin ja hyväksyttävissä olevan riskitason suhteena. Tätä suhdetta tai logaritmista erotusta kutsutaan turvallisuuden eheystasoksi.

Yleensä koko TLJ:tä ei kokonaisuudessaan suunnitella alusta lähtien, vaan sovellussuunnittelu tehdään valmiille perusjärjestelmälle tai alustalle. Perusjärjestelmä on tavalla tai toisella todettu täyttävän sovellussuunnittelulle vaadittavat ominaisuudet. Yleisesti sanottuna sovellussuunnittelija ostaa kaupan hyllyltä tietyn TETin omaavan automaatiojärjestelmän alustan ja toteuttaa sille sovelluksensa. Tietääkseen automaatiojärjestelmältä vaadittavan TETin, sovellussuunnittelijan on määrättävä jokaisen merkittävän vaarallisen tapahtuman riski ja pääteltävä minkä suuruiseksi riski tulisi vähentää. Väliin jäävää osuutta vaarallisen tapahtuman riskin ja siedettävän riskin välillä kutsutaan riskinvähennykseksi. Kun riskin vähennys asetetaan diskreetiksi logaritmiseksi luvuksi, saadaan tarvittava vähennys turvallisuuden eheystasoina.

Turvatoimintokokonaisuus voidaan jakaa osatoimintoihin, joihin kohdistuva turvallisuuden eheysvaatimus on johdettavissa koko turvatoiminnon eheydestä. Osatoiminnot voivat ovat sovellusspesifistisiä tai yleistarkoituksellisia. Jälkimmäisiä ovat mm. laitespesifistiset toiminnot: anturi-, toimielin-, tiedonsiirto-, käyttöjärjestelmä-, järjestelmäkutsu- ja ajuritoiminnot.

Vaarallisen tapahtuman riski esitetään yleensä tulona:

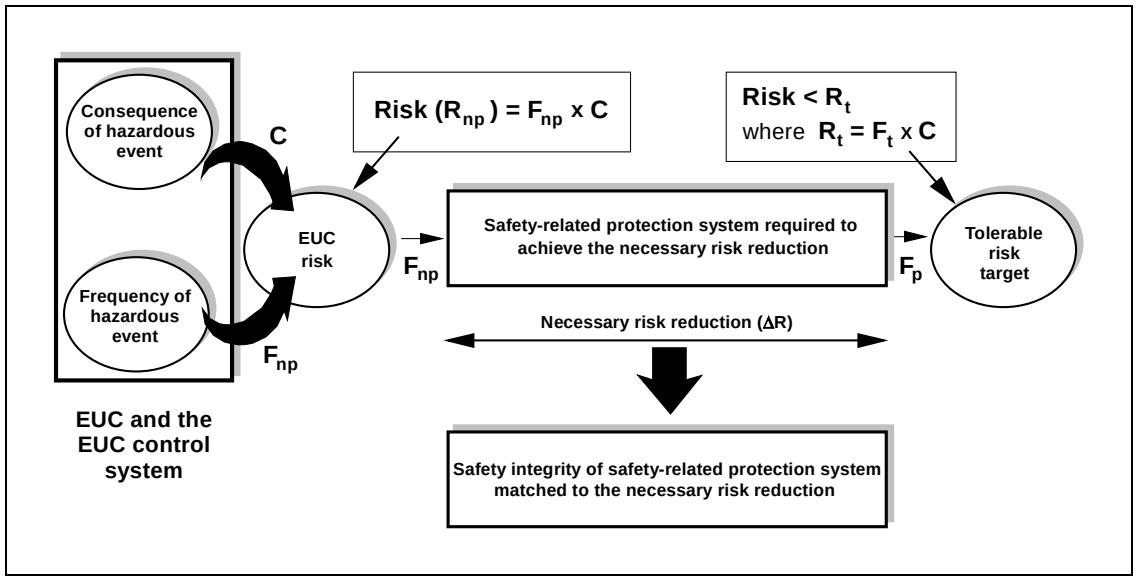
$$R = F_{np} \; x \, C, \tag{1}$$

missä R = riski, C = vaarallisen tapahtuman seuraus, ja F_{np} = vaarallisen tapahtuman todennäköisyys tai taajuus.

Vaaralliselle tapahtumalle määritetään tarpeellinen riskin vähennys, mikä voidaan tehdä joko kvantitatiivisesti ja/tai kvalitatiivisesti. Riskiä voi vähentää pienentämällä vaaralli-

sen tapahtuman seurausta. Yleensä kuitenkin pienennetään vaarallisen tapahtuman todennäköisyyttä. TLJ:t vähentävät riskiä turvatoimintojen kautta. Siten jokaiselle turvatoiminnolle määritetään turvallisuuden eheyden vaatimukset.

Kuvassa B4 on esitetty riskin vähennyksen yleiset periaatteet. Ohjattavan kohteen riski (EUC risk) tarkoittaa riskiä, joka tulee ohjattavasta laitteistosta tai sen vuorovaikutuksesta ohjausjärjestelmän ja operaattorin kanssa. Siedettävä riski (tolerable risk target) on määritelty riskinä, joka hyväksytään tietyssä yhteydessä yhteiskunnan sen hetkisten arvojen mukaan. Se saattaa vaihdella eri sovellussektoreilla, maissa ja käyttäjäyhtiöissä.



Kuva B4. Esimerkki turvallisuuden eheyden jakamisesta turvallisuuteen liittyvälle järjestelmälle. EUC = Equipment Under Control (EN-IEC 61508).

Kun turvatoimintojen eheyden määrittely on edennyt riittävästi, TLJ:n jokaisen turvatoiminnon turvallisuuden eheyden vaatimukset tulee määrittää oheisten taulukoiden B1 ja B2 mukaan tietyille TET:eille. Taulukon valinta riippuu siitä onko tavoitteellisen vikaantumismitan parametri joko

- harvojen vaateiden toimintatapaa varten, jolloin kyseessä on keskimääräinen epäonnistumisen todennäköisyys vaadittaessa, tai
- tiheiden vaateiden tai jatkuvan toiminnan toimintatapaa varten, jolloin kyseessä on vaarallisen vikaantumisen todennäköisyys tuntia kohti.

B3.1 Kvantitatiivinen menetelmä

Kuva antaa esimerkin kuinka yksittäisen turvallisuuteen liittyvän suojaustoiminnon tavoitteellinen turvallisuuden eheys lasketaan. Sellaiselle tapaukselle minimi riskin vähennykseen pääsemiseksi

$$PFD_{avg} \leq F_t / F_{np} \qquad (2)$$

missä, PFD_{avg} on turvallisuuteen liittyvän suojaustoiminnon keskimääräinen vikaantumistodennäköisyys, mikä on turvallisuuden eheyden vikaantumismitta harvojen vaateiden toimintatavan turvallisuuteen liittyville järjestelmille (ks. Taulukko).

F_t on siedettävän riskin taajuus

F_{np} on turvallisuuteen liittyvään suojaustoimintoon kohdistuva vaadetaajuus.

C on vaarallisen tapahtuman seuraus;

F_p on riskitaajuus suojausosien ollessa paikallaan.

TET:n saamiseksi, kun seuraus C pysyy vakiona, menetellään seuraavasti (ks. Kuva) siinä tapauksessa, että koko tarpeellinen riskin vähennys saavutetaan yhdellä turvallisuuteen liittyvällä suojausjärjestelmällä, jonka täytyy vähentää vaarataajuutta vähintään F_{np}:stä F_t:hen:

1. Määritä ohjattavan kohteen riskin taajuusosa ilman suojaavia osia, F_{np}
2. Määritä seuraus C ilman suojaavia osia.
3. Määritä turvallisuuteen liittyvän suojaustoiminnon vikaantumisen todennäköisyys vaadittaessa PFD_{avg} tarpeellisen riskin vähentämisen ΔR täyttämiseksi. Seurauksen ollessa vakio kuvatussa tapauksessa, $PFD_{avg} = F_t/F_{np} = \Delta R$
4. Määritä taulukosta (Taulukko) suureelle $PFD_{avg} = (F_t/F_{np})$ turvallisuuden eheyden taso (esim., jos $PFD_{avg} = 10^{-2} \ldots 10^{-3}$, TET = 2).

Menettely tulee toistaa jokaiselle suojaustoiminnolle.

B3.2 Kvalitatiiviset menetelmät

Riskigraafi on kvalitatiivinen menetelmä turvallisuuteen liittyvän järjestelmän (TLJ) turvallisuuden eheyden tason (TET) määrittämiseksi ohjattavan kohteen ja sen ohjausjärjestelmän riskitekijöiden perusteella.

Kun käytetään kvalitatiivista menetelmää, otetaan käyttöön yksinkertaisuuden vuoksi joukko parametreja, jotka kuvaavat yhdessä vaarallista tilannetta joka syntyy, kun TLJ vikaantuu tai ei ole käytössä. Neljästä parametrijoukosta valitaan kustakin yksi parametri ja valitut parametrit yhdistetään TLJ:n TET:n päättämiseksi.

Riskigraafi ovat EN-IEC 61508:n informatiivisissa osissa. Toisin sanoen sen tilalle voi kehittää jotain vastaavia ko. sovellukseen paremmin sopivia menetelmiä. Seuraavassa esiteltävä riskigraafi on toistaiseksi vaikuttanut käyttökelpoiselta prosessiteollisuuteen. Riskigraafeja on henkilö-, ympäristö- ja taloudellisilta vahingoilta suojaamiseen.

Esitettävä yksinkertaistettu menetelmä perustuu seuraavaan yhtälöön:

$$R = f \times C,$$ missä:

- R on riski kun ei ole turvallisuuteen liittyvää järjestelmää
- f on vaarallisen tapahtuman taajuus kun ei ole turvallisuuteen liittyvää järjestelmää
- C on vaarallisen tapahtuman seuraus (tässä henkilöiden terveydelle ja turvallisuudelle)

Tässä tapauksessa vaarallisen tapahtuman taajuuden f katsotaan koostuvan seuraavista tekijöistä:

- vaarallisella alueella oleskelun taajuus ja alttiina olon aika
- vaarallisen tapahtuman välttämismahdollisuus
- vaarallisen tapahtuman todennäköisyys ilman turvallisuuteen liittyvää järjestelmää

Näin saadaan seuraavat neljä riskiparametriä:

- vaarallisen tapahtuman seuraus C
- vaarallisella alueella oleskelun taajuus ja alttiina olon aika F
- vaarallisen tapahtuman välttämistodennäköisyys P
- ei-toivotun tapahtuman todennäköisyys W

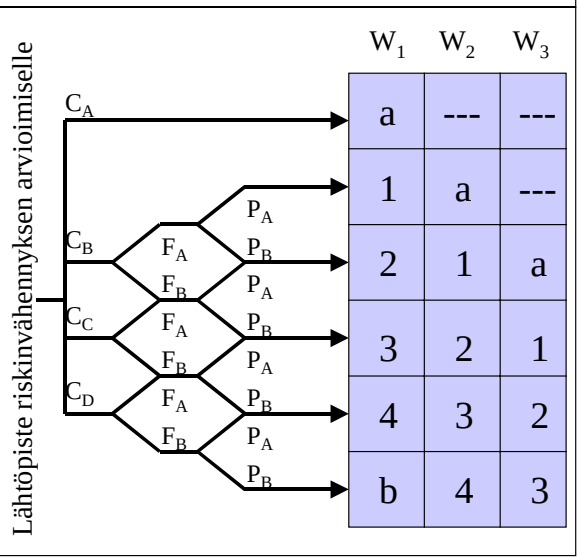
Kuva B5 esittää yleistä riskigraafia, joka on sovellettavissa eri toimialoille sopivilla parametrivalinnoilla C, F, P ja W.

VAHINGON LAAJUUS, C
Vaarasta todennäköisesti seuraava tapaturmien keskimääräinen lukumäärä. Tämä lasketaan keskimääräisellä henkilöluvulla alttiilla alueella, kun alueella oleskellaan ottaen huomioon vaarallisen tapahtuman aiheuttama haavoittuvuus.

OLESKELU, F
Todennäköisyys, että alttiilla alueella oleskellaan. Tämä määrätään laskemalla se osa ajasta, jolloin alueella oleskellaan.

VAARAN VÄLTTÄMINEN, P
Todennäköisyys, että alttiina olevat henkilöt pystyvät välttämään vaaran, jos TLJ ei toimi vaadittaessa.
Tämä riippuu siitä, että on riippumattomia alttiina olevien henkilöiden varoitusmenetelmiä ja käsikäyttöisiä menetelmiä vaaran estämiseen tai pakomahdollisuuksia.

VAADETAAJUUS, W
Vaarallisen tapahtuman lukumäärä vuodessa, jos ei ole TLJ:tä. Tämä voidaan määrittää huomioimalla kaikki vikaantumiset, jotka voivat johtaa ko. vaaraan ja laskemalla niiden esiintymisen kokonaistaajuus.



Kuva B5. Geneerinen riskigraafi turvallisuuden eheystasojen (1, 2, 3 ja 4) määrittämiseksi. Riskiparametrit C, F ja P määritellään sovellusalakohtaisesti tai sektoristandardeissa. ”a” tarkoittaa ettei erityisiä turvallisuusvaatimuksia tarvita, ”b” tarkoittaa ettei yksittäinen turvallisuuteen liittyvä järjestelmä riitä.

Liitteen B lähdeluettelo

ANSI/ISA-S84.01, Standard 1996. Application of safety instrumented systems to the process industires. Instrument Society of America, ISA, February 15, 1996.

ATU, Suomen Automaation Tuki Oy. 2001. Laatu automaatiossa. Parhaat käytännöt. Tommila, T. (ed.). Helsinki: Suomen Automaatioseura ry. 245 s. ISBN 952-5183-12-2.

EN-IEC 61508, Standardi. 2001. Functional safety of programmable electronic systems: International Electrotechnical Commission. Parts: 1–7.

IEC 61511, Standardi. 2001. Functional safety: Safety Instrumented Systems for the process industry sector. Draft, CDV.

Julkaisija

Vuorimiehentie 5, PL 2000, 02044 VTT
Puh. (09) 4561
Faksi (09) 456 4374

Julkaisun sarja, numero ja raporttikoodi

VTT Tiedotteita 2151
VTT–TIED–2151

Tekijä(t)
Harju, Hannu

Nimeke

Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi Osa 1

Tiivistelmä

Ohjelmistojen käyttäminen kriittisiin sovelluksiin on jatkuvassa kasvussa. Päinvastoin kuin laitteistoviat, ohjelmistoviat ovat systemaattisia ja ohjelmistovirheet voivat piileksiä pitkiä aikoja ennen paljastumistaan. Tässä tiedotteessa tarkastellaan ensiksi yleisesti eräitä ohjelmiston luotettavuuteen vaikuttavia tärkeimpiä tekijöitä, sitten erityisteemoina eheystason hyödyntämistä, valmisohjelmistojen luotettavuutta ja ohjelmiston luotettavuustakuita.

Monet turvallisuuskriittisten ohjelmistojen kehitysstandardit tukeutuvat käsitteeseen turvallisuuden eheystaso. Se on todennäköisyysmitta sille, että turvatoiminto täyttää sille asetetut turvallisuusvaatimukset. Turvallisuuden eheystasomenettelyllä on ilmeiset etunsa mutta myös ongelmansa, jotka vaikeuttavat menettelyn hyödyntämistä käytännössä. Tiedote sisältää lyhyen viitestandardien tarkastelun tavoitteena selvittää lähestymistavan hyödyntämismahdollisuudet muille luotettavuusattribuuteille kuin turvallisuus.

Kaupallisesti saatavilla oleva ohjelmistokomponentti, jota usein kutsutaan COTSiksi, on komponentti, joka on tarkoitettu laajaan teollisuus- ja sovelluskäyttöön. COTSeista koostuvat sovellukset ovat aikaa myöten edullisempia kuin varta vasten kehitetyt järjestelmät. COTSeilla on laajat käyttökokemukset, mutta erityisesti turvallisuuteen liittyvien järjestelmäsovellusten haittana on käyttökokemustietojen vaikea saatavuus, mikä on edellytys vakuuttautumisessa COTS-pohjaisista järjestelmistä. Tutkimuksessa keskitytään luotettavuuden arviointiin sekä kolmannen osapuolen että luotettavuusominaisuuksien näkökulmasta.

Hyvän ohjelmiston kehitysprosessin ongelma on siinä, ettei se takaa hyvää ohjelmistoa. Erilaiset tuotteen arviointimenetelmät (mm. testit ja formaalit verifioinnit) eivät myöskään takaa hyvää ohjelmistoa. Ohjelmiston laadun evaluointi kohdistuu henkilö-, tuote- ja prosessiarviointiin. Niistä julkaisu keskittyy tuotteenarviointiin kolmesta hyvin erilaisesta näkökulmasta: sertifiointi pisteytysmenetelmällä, ohjelmiston testauksiin perustuvalla luotettavuustekniikalla ja riskienhallinnan tekniikalla.

Avainsanat
software dependability, safety integrity levels, reliability scoring, software reliability engineering, risk management process

Toimintayksikkö
VTT Tuotteet ja tuotanto, Tekniikantie 12, PL 1301, 02044 VTT

ISBN	Projektinumero
951–38–6062–0 (nid.) 951–38–6063–9 (URL: http://www.inf.vtt.fi/pdf/)	A2SU00140

Julkaisuaika	Kieli	Sivuja	Hinta
Elokuu 2002	Suomi, engl. abstr.	114 s. + liitt. 15 s.	C

Toimeksiantaja(t)
Tekes, ABB Industry, Fortum, Teollisuuden Voima Oy, VTT

Avainnimeke ja ISSN	Myynti:
VTT Tiedotteita – Research Notes 1235–0605 (nid.) 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)	VTT Tietopalvelu PL 2000, 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374

Published by

Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
Phone internat. +358 9 4561
Fax +358 9 456 4374

Series title, number and report code of publication

VTT Research Notes 2151
VTT–TIED–2151

Author(s)
Harju, Hannu

Title

Cost-effective design and assessment of dependable software
Part 1

Abstract

Software is increasingly being used in critical applications. Unlike most hardware failures, software failures are systematic and software faults may lie hidden for a long time before being revealed. This publication first introduces some affects that impact software reliability, and then three different aspects of software dependability: utilising integrity levels, dependability of COTS software, and warranty of software dependability.

Many standards for the development of safety-critical software and systems introduce the notion of safety integrity levels. It is a measure of the required likelihood that a safety function achieves its safety requirements. The approach of the safety integrity level has its obvious advantages, but also problems which make it inconvenience for utilising in practice. This publication contains a brief review of some widely referenced standards, to consider a context for utilising the approach for other dependability attributes than safety.

A commercially available (frequently called COTS) software component is one that has been developed for use in a variety of industries and applications. The system applications developed with the COTS software component are expected to be cheaper over the lifetime of the application than the custom designed plant application. The COTS system has a wider experience base. The disadvantages, especially for safety related systems, is that it may be difficult to get enough information to make a convincing acceptance argument for the COTS -software based application. The COTS software and systems constitued from them is considered from third parties viewpoint and from dependability features as design, security, maintainability and concentrated failures.

A problem of good software development processes is that they do not guarantee good software. Different forms of product assessment (testing as well as techniques such as formal verification) are developed, but also they do not guarantee good software. Evaluation of software quality consists of three parts: personal, product and process evaluations. All of them is also needed for a dependability evaluation. The publication is concentrated in product certification from three different viewpoints: reliability scoring, software reliability engineering, risk management process.

Keywords
software dependability, safety integrity levels, reliability scoring, software reliability engineering, risk management process

Activity unit
VTT Industrial Systems, Tekniikantie 12, P.O.Box 1301, FIN–02044 VTT, Finland

ISBN
951–38–6062–0 (soft back ed.)
951–38–6063–9 (URL: http://www.inf.vtt.fi/pdf/)

Project number
A2SU00140

Date	Language	Pages	Price
August 2002	Finnish, Engl. abstr.	114 p. + app. 15 p.	C

Commissioned by
Tekes, ABB Industry, Fortum, Teollisuuden Voima Oy, VTT

Series title and ISSN
VTT Tiedotteita – Research Notes
1235–0605 (soft back edition)
1455–0865 (URL: http://www.inf.vtt.fi/pdf/)

Sold by
VTT Information Service
P.O.Box 2000, FIN–02044 VTT, Finland
Phone internat. +358 9 456 4404
Fax +358 9 456 4374

VTT TIEDOTTEITA – RESEARCH NOTES

VTT TUOTTEET JA TUOTANTO – VTT INDUSTRIELLA SYSTEM – VTT INDUSTRIAL SYSTEMS

2050 Kotikunnas, Erkki & Heino, Perttu. Turvallisen prosessilaitoksen suunnittelu. STOPHAZ-projektissa syntyneet työkalut. 2000. 44 s.

2058 Konola, Jari. Kunnossapidon tietojärjestelmä käyttövarmuustiedon lähteenä Suomen paperi- ja selluteollisuudessa. 2000. 25 s.

2061 Välisalo, Tero & Rouhiainen, Veikko. Luotettavuusjohtaminen työkoneteollisuudessa. 2000. 43 s. + liitt. 15 s.

2063 Tonteri, Hannele, Vatanen, Saija & Kuuva, Markku. Työkoneiden käytön jälkeisen käsittelyn suunnittelu. 2000. 32 s. + liitt. 4 s.

2064 Tonteri, Hannele, Vatanen, Saija & Kuuva, Markku. Design for end-of-life treatment of work machines. 2000. 32 p. + app. 4 p.

2066 Harju, Hannu. Ohjelmiston luotettavuuden kvalitatiivinen arviointi. 2000. 111 s.

2067 Baumont, Geneviève, Wahlström, Björn, Solá, Rosario, Williams, Jeremy, Frischknecht, Albert, Wilpert, Bernhard & Rollenhagen, Carl. Organisational factors. Their definition and influence on nuclear safety. Final raport. 2000. 65 p.

2077 Solin, Jussi (ed.). Plant life management (XVO). Report 1999. 2001. 68 p. + app. 3 p.

2098 Parikka, Risto, Ahlroos, Tiina, Halme, Jari, Miettinen, Juha, Salmenperä, Pekka, Lahdelma, Sulo, Kananen, Markku & Kantola, Petteri. Monitorointi ja diagnostiikka. 2001. 55 s.

2115 Luoma, Tuija, Mattila, Inga, Nurmi, Salme, Ilmén, Raija, Heikkilä, Pirjo, Salonen, Riitta, Sikiö, Teija, Lehtonen, Mari & Anttonen, Hannu. Elektroniikka- ja kemianteollisuuden suojavaatteet. Sähköstaattiset ominaisuudet ja käyttömukavuus. 2001. 92 s. + liitt. 12 s.

2117 Malm, Timo, Hämäläinen, Vesa & Kivipuro, Maarit. Paperiteollisuuden rullankäsittelyn turvallisuus ja luotettavuus. 2001. 68 s. + liitt. 12 s.

2140 Reiman, Teemu & Oedewald, Pia. The assessment of organisational culture. A methodological study. 2002. 42 p.

2148 Aaltonen, Pertti, Bojinov, Martin, Helin, Mika, Kinnunen, Petri, Laitinen, Timo, Muttilainen, Erkki, Mäkelä, Kari, Reinvall, Anneli, Saario, Timo & Toivonen, Aki. Facts and views on the role of anionic impurities, crack tip chemistry and oxide films in environmentally assisted cracking. 2002. 68 p. + app. 21 p.

2149 Hemilä, Jukka. Information technologies for value network integration. 2002. 97 p. + app. 1 p.

2150 Pöyhönen, Ilpo, Kylmälä, Kaarle, Harju, Hannu, Kemppainen-Kajola, Pia, Kuhakoski, Kalle, Spankie, Greig & Ventä, Olli. Vaatimukset ohjelmistoa sisältäville lääkintälaitteille. Hallinta ja menetelmät vaatimustenmukaisuuden osoittamiseksi. 2002. 135 s. + liitt. 40 s.

2151 Harju, Hannu. Kustannustehokas ohjelmiston luotettavuuden suunnittelu ja arviointi. Osa 1. 2002. 114 s. + liitt. 15 s.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU PL 2000 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374	VTT INFORMATIONSTJÄNST PB 2000 02044 VTT Tel. (09) 456 4404 Fax (09) 456 4374	VTT INFORMATION SERVICE P.O.Box 2000 FIN–02044 VTT, Finland Phone internat. + 358 9 456 4404 Fax + 358 9 456 4374

ISBN 951–38–6062–0 (nid.)
ISSN 1235–0605 (nid.)
ISBN 951–38–6063–9 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)