2066

VTT TIEDOTTEITA

VTT TIEDOTTEITA

Hannu Harju

Ohjelmiston luotettavuuden kvalitatiivinen arviointi

VALTION TEKNILLINEN TUTKIMUSKESKUS

ESPOO 2000

VTT TIEDOTTEITA – MEDDELANDEN – RESEARCH NOTES 2066

Ohjelmiston luotettavuuden kvalitatiivinen arviointi

Hannu Harju
VTT Automaatio

VALTION TEKNILLINEN TUTKIMUSKESKUS
ESPOO 2000

ISBN 951–38–5766–2 (nid.)
ISSN 1235–0605 (nid.)

ISBN 951–38–5767–0 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

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

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

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

VTT Automaatio, Teollisuusautomaatio, Tekniikantie 12, PL 1301, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 6752

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

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

Toimitus Leena Ukskoski

Otamedia Oy, Espoo 2000

Harju, Hannu. Ohjelmiston luotettavuuden kvalitatiivinen arviointi [The qualitative assessment of software dependability]. Espoo 2000, Valtion teknillinen tutkimuskeskus, VTT Tiedotteita – Meddelanden – Research Notes 2066. 111 s.

Avainsanat software safety integrity, software reliability, software availability, software maintainability, software fault mechanisms, software dependability requirements validation

Tiivistelmä

Tekniset järjestelmät tulevat jatkuvasti yhä enemmän riippuviksi ohjelmistoista. Ohjelmistopohjaiset turvatoimintoja toteuttavat turvallisuuteen liittyvät järjestelmät pohjautuvat tiukkoihin vaatimuksiin ohjelmiston turvallisuuden eheydestä. Niille on olemassa joukko menetelmiä ja tekniikoita, joilla voidaan arvioida ja suunnitella ohjelmiston turvallisuuden eheyttä. Tarve hyödyntää näitä menetelmiä ja tekniikoita on hyvin tiedostettu, mutta korkeitten kustannusten takia niitä ei ole käytetty vähemmän kriittisten ohjelmistojärjestelmien rakentamisessa.

Tässä julkaisussa kuvataan ohjelmiston luotettavuuden kvalitatiivinen kelpoistusmenetelmä, jonka tarkoituksena on arvioida ja ohjata luotettavuusominaisuuksia ohjelmistotuotannon elinkaaren vaiheissa. Ohjelmiston luotettavuusominaisuuksilla tarkoitetaan neljää luotettavuusattribuuttia (toimintavarmuus, käytettävyys, ylläpidettävyys ja turvallisuus), virhemekanismeja (virhe, virhetilanne ja virhetoiminto) sekä keinoja virhemekanismien purkamiseksi tietyn luotettavuuden saavuttamiseksi.

Ohjelmiston luotettavuusvaatimusten kelpoistamismenetelmän kehittämisessä on otetava huomioon muut ohjelmiston virheettömyyteen ja oikeaan suoritukseen tähtäävät menetelmät (luku 4). Menetelmä toteutetaan tietyillä vaara-analyysitekniikoilla (luku 5), jotka sopivilla kriteerivalinnoilla (luku 6) yhdistetään kustannustehokkaaksi menetelmäksi Stérna (luku 7). Stérnaa koekäytettiin yhteistyöyritysten Honeywell-Measurexin ja Neles Automationin ohjelmistoprojekteissa.

Harju, Hannu. Ohjelmiston luotettavuuden kvalitatiivinen arviointi [The qualitative assessment of software dependability]. Espoo 2000, Technical Research Centre of Finland, VTT Tiedotteita – Meddelanden – Research Notes 2066. 111 p.

Keywords software safety integrity, software reliability, software availability, software maintainability, software fault mechanisms, software dependability requirements validation

Abstract

In recent years, many engineering systems have become increasingly dependent on software. Software based safety related systems which perform safety functions put stringent requirements on the safety integrity of the software involved. Therefore there exists a set of methods and tehcniques that can be used to assess and design safety integrity of software. A need to utilize these methods for less critical items of software is well known, but there isn't reasonably cost-effective methods for software project to use in industrial areas as nuclear power, trafic and medical.

At Chapter seven, a method for software dependability requirements validation to assess and control attributes of software dependability features is described. The dependability features are the four attributes (reliability, availability, maintainability and safety), fault mechanisms (fault, error and failure), and means to remedy fault mechanisms for a level of given dependability.

In developing the method of software dependability requirements validation the other methods aimed to improve correctness of software have to be considered (Chapter 4). The method will be carried on by integrating a set of techniques of specified safety analysis (described at Chapter 5). Techniques will be selected by appropriate criterias (Chapter 6) for conducting cost-effective dependability validation of the software called Stérna. Stérna was tested at software case projects of co-operation company of Honeywell-Measurex and Neles Automation.

Alkusanat

Tämä julkaisu esittelee luotettavuuden laadullista eli kvalitatiivista arviointia ohjelmistokehityksen elinkaarivaiheissa. Lähtökohdat on otettu perinteisistä riskianalyysin menetelmistä. Niistä on integroitu ohjelmiston luotettavuusvaatimusten kelpoistusmenetelmä.

Menetelmä laadittiin VTT Automaatiossa tutkimusprojektissa EKS, *kokonaisluotettavuuden integroitu analyysimenetelmä,* joka kuului Tekesin tutkimusohjelmaan ETX, *Elektroniikka tietoyhteiskunnan palveluksessa*. EKS-projekti toteutettiin yhteistyössä yritysten Honeywell oyj Measurex ja Neles Automation oyj kanssa. Sen koordinointia varten perustetussa johtoryhmässä olivat Tekesin edustajana Kari Rintala, osallistuvien yritysten edustajina Kari Hartikainen Neles Automationista ja Juha Silander Honeywell-Measurexista sekä projektin tutkimuksen edustajana Olli Ventä VTT Automaatiosta. Heidän lisäkseen haluan lausua parhaat kiitokset Sami Hakuliselle, Lauri Mustoselle, Pia Kemppainen-Kajolalle, Kalle Kuhakoskelle ja Tony Rosquistille, jotka ovat kannustavalla työpanoksellaan osallistuneet tutkimukseen.

VTT Automaatio
Teollisuusautomaatio
Espoo 15.02.2000

Hannu Harju

Sisällysluettelo

Tiivistelmä 3

Abstract 4

Alkusanat 5

Lyhenteet 8

1. Johdanto 9

2. Tutkimusmenetelmä 11

3. Ohjelmiston luotettavuusominaisuudet 14
   3.1 Käsitteet 14
   3.2 Ohjelmiston virhemekanismit 16
      3.2.1 Virhetoiminta 17
      3.2.2 Virhetilanteet 18
      3.2.3 Virheet 19
   3.3 Luotettavuutta kasvattavat keinot 21
      3.3.1 Virheiden välttäminen ja poistaminen 22
      3.3.2 Virhesietoisuus 23
      3.3.3 Virheiden ennustaminen 26
   3.4 Luotettavuusattribuutit 28
      3.4.1 Toimintavarmuus ja käytettävyys 28
      3.4.2 Kriittinen ylläpidettävyys 29
      3.4.3 Ohjelmiston turvallisuus 31
   3.5 Luotettavuusvaatimusten asettaminen 33
      3.5.1 Vaatimusasettelun strategia 33
      3.5.2 Luotettavuusprofiili 34

4. Luotettavuuden asettamat vaatimukset ohjelmistotekniikalle 42
   4.1 Ohjelmiston vaatimusmäärittely 42
   4.2 Ohjelmistosuunnittelu 44
   4.3 Ohjelmiston toteutusvaihe 47
   4.4 Ohjelmiston ylläpito 48
   4.5 Tarkistukset 49

5. Analyysimenetelmät ja -tekniikat 52
   5.1 Luotettavuuden arvioinnin puitteet 53
   5.2 Luotettavuusvaatimusten kelpoistaminen 56

5.2.1 Alustava vaara-analyysi ... 56
5.2.2 Vaara-analyysi ... 57
5.2.3 Yhteisvika-analyysi ... 58
5.3 Luotettavuusvaatimusten kelpoistustekniikat ... 59
5.3.1 Ohjelmiston vika-, vaikutus- ja kriittisyysanalyysi ... 59
5.3.2 Ohjelmiston vikapuuanalyysi ... 64
5.3.3 Poikkeamatarkastelu ... 66
5.3.4 Tapahtumapuuanalyysi ... 67
5.3.5 Ohjelmiston oikopolkuanalyysi ... 69
5.3.6 Petriverkot ... 71
5.4 Muut luotettavuusanalyysit ... 73

6. Analyysikriteerit ... 76
6.1 Informaatiokriteeri ... 77
6.2 Resurssikriteeri ... 78
6.3 Yhteensopivuuskriteeri ... 80

7. Ohjelmiston luotettavuuden kelpoistusmenetelmä ... 82
7.1 Menetelmän periaatteet ... 83
7.2 Avainlauseet ... 84
7.3 Analyysin suunnittelu ... 86
7.3.1 Tarkastelukohde ... 87
7.3.2 Järjestelmätasojen vaara-analyysit ja kriittisyyslista ... 88
7.3.3 Luotettavuusprofiili ... 91
7.3.4 Luotettavuusanalyysin suunnitelma ... 91
7.4 Vaatimusmäärittelyn luotettavuusanalysointi ... 92
7.5 Arkkitehtuurisuunnittelun luotettavuusanalyysi ... 95
7.6 Suunnittelun luotettavuusanalyysi ... 99
7.7 Toteutuksen luotettavuusanalyysi ... 101

8. Yhteenveto ja johtopäätökset ... 105

Lähdeluettelo ... 108

Lyhenteet

ETA	Tapahtumapuuanalyysi (Event Tree Analysis)
FMEA	Vika- ja vaikutusanalyysi (Failure Mode and Criticality Analysis)
FMECA	Vika-, vaikutus- ja kriittisyysanalyysi (Failure Mode, Effect and Criticality Analysis)
FTA	Vikapuuanalyysi (Fault Tree Analysis)
HAZOP	Poikkeamatarkastelu (Hazard and Operability Study)
HSIA	Laitteiston ja ohejlmiston vuorovaikutusanalyysi (Hardware Software Interaction Analysis)
MTTF	Keskimäärinen vioittumisaika (Mean Time to Failure)
MTTR	Keskimääräinen korjausaika (Mean Time to Repair)
PHA	Alustava vaara-analyysi (Priliminary Hazard Analysis)
RPN	Riskiluku (Risk Priority Number)
SRHA	Ohjelmistovaatimusten turvallisuusanalyysi (Software Requirements Hazard Analysis)
SSA	Ohjelmiston oikopolkuanalyysi (Software Sneak Analysis)

1. Johdanto

Luotettavuudesta puhutaan silloin, kun minkä tahansa kohteen virheetön toiminta on hyödyntäjälle tärkeää. Mikään muu teknologia ei ole saavuttanut niin laajaa rynnäkköä hyödyntäjän käsiin kuin ohjelmisto. Minkään muun luotettavuudesta ei puhuta niin paljon kuin ohjelmiston. Vaikka ohjelmistoja tarvitaan yhä enemmän kaikilla yhteiskunnan aloilla ja jopa ihmisen terveys ja henki voivat vaarantua niiden virhetoiminnasta, luotettavuus on kehityksessä yhä edelleen jälkeenjäänein ohjelmiston kaikista lukuisista laatuominaisuuksista. Mekaniikalle ja elektroniikalle luotettavuuden arvioiminen sekä laadullisesti että määrällisesti on täsmällistä tiedettä, vaikka.ohjelmistojen luotettavuuden arviointitapojen kehittämiseen panostetaan vuosittain huomattavilla rahamäärillä.

Mitä tarkoitetaan luotettavuudella? Jos ohjelmisto on virheetön, onko se myös luotettava, käytettävä ja turvallinen? Milloin ohjelma on ylläpidettävä? Miten nämä käsitteet eroavat toisistaan? Kun luotettavuus määritellään virheettömyytenä tietyissä olosuhteissa ja ympäristössä [Musa et al. 1987], miten luotettavuustarkastelut eroavat virheettömyyden tarkasteluista? Mitä ovat ohjelmiston luotettavuuden tarkastelumenetelmät ja -tekniikat – laadulliset ja määrälliset? Miten niitä hyödyntää esimerkiksi virhesietoisen ohjelmistojärjestelmän suunnittelussa, testitapausten valinnassa tai oikeellisuuden arvioimisessa? Kuormittavatko ne ainoastaan ohjelmistokehitystä – riittävätkö testit ja katselmukset?

Luotettavuus on keskeinen laatuominaisuus. Se liitetään olennaiseksi osaksi kaikkiin kehitettyihin laaturakenteisiin [McCabe 1976, ISO/IEC 9126 1991] joko ominaisuutena tai kriteerinä. Sen parantamiseksi on synnytetty laatujärjestelmiä. Se rakentuu monista osatekijöistä kaikissa ohjelmistotuotannon elinkaaren vaiheissa sekä niiden todennuksissa ja kelpoistuksissa. Ja luotettavuuden ongelmat juontuvat kaikista näistä vaiheista ja tehtävistä.

Ohjelmiston todennusmenetelmistä testaaminenkin on ongelmallista. Se on yhä edelleen tärkein oikeellisuuden osoittamistapa, mutta on alun perin suunnattu laitteistoille, eikä modernia ohjelmiston luotettavuustestausta ole ilmestynyt näköpiiriin. Määrälliset luotettavuustestaukset ovat kohdentuneet ensi sijassa vain hyvin korkeaa ohjelmiston luotettavuutta vaativiin kohteisiin. Samoin kuin riskianalyysin tekniikat, vaara- tai turvallisuusanalyysit ovat suuntautuneet vain hyvin korkeaa turvallisuutta edellyttäviin laitteisiin ja prosesseihin.

Tämä julkaisu perustuu tutkimukseen, johon lähdettiin muutamasta olettamuksesta. Ensinnäkin oletettiin, että riskianalyysitekniikat sopivat kustannustehokkaasti vähemmänkin kriittisille ohjelmistoaloille, joissa tiedostetaan lyhytaikaisenkin ohjelmiston virhe-

toiminnan aiheuttavan kiusallisia toimintahäiriöitä, käyttökeskeytyksiä ja suuritöistä ylläpitoa. Toiseksi oletettiin, että laadullisillakin tarkasteluilla kyetään arvioimaan riittävän kattavasti ohjelmiston luotettavuutta. Kolmanneksi lähtökohdaksi valittiin kokonaisvaltaisuus, johon haluttiin pyrkiä laadullisilla tarkasteluilla sekä suuntaamalla että ohjeistamalla ohjelmistotuotantoa kohti luotettavia ratkaisuja.

Tutkimuksessa lähdetään myös siitä, että ohjelmistosuunnittelussa on tiedostettu riittävän selkeästi se, miten luotettavia, so. toimintavarmoja, käyttövarmoja, turvallisia ja ylläpidettäviä, rakennettavan ohjelmiston toimintojen tulisi olla. Rakennettiin ohjelmisto millä tavalla tahansa, toimintoihin kohdistuvat luotettavuusvaatimukset tulisi asettaa jossakin sopivassa, mieluimmin varhaisessa elinkaarivaiheessa. Oli elinkaari toteutettu inkrementaalisti/enenevästi, iteroivasti/kertautuvana, spiraalisti tai integroituna modernimpiin ohjelmistoprosesseihin, edellytys luotettavuusvaatimuksista ei ole liioiteltua sikälikään, että ohjelmoijat ovat tottuneet puurtamaan koodia ulkoisten tavoitteiden saattelemina.

Luotettavuuteen liittyy aina tietyntasoinen kriittisyys, mikä ilmaistaan vaatimuksissa. Ilman kriittisyyttä ohjelmistotoimintojen hyödyntäjä ei puhuisi luotettavuudesta. Tästä ja tarpeesta luotettavuuden osoittamiseen seuraa, että luotettavuustarkasteluissa on kyse luotettavuusvaatimusten kelpoistamisesta. Kelpoistamisellahan tarkoitetaan alkuperäisten vaatimusten toteutumisen osoittamista.

Tutkimuksessa kehitettiin ohjelmiston luotettavuusvaatimusten kvalitatiivinen kelpoistusmenetelmä. Kehittämisessä tarkasteltiin aluksi sitä, mistä ohjelmiston luotettavuus rakentuu, mistä luotettava ohjelmistotuotanto koostuu, millaisia luotettavuuden tarkastelutapoja on olemassa sekä millaisia kriteereitä tarkastelutapojen valintaan kohdistetaan.

Tässä julkaisussa menetelmäksi kutsutaan mitä hyvänsä tietyn tehtävän toteuttavaa toimintatapaa tai prosessia. Myös tekniikka voi olla menetelmä, mutta tässä yhteydessä tietty menetelmä joko kokonaan toteutetaan tietyllä tekniikalla tai tekniikka on osa kokonaisuutta. Niinpä esimerkiksi standardinmukaiset tai tietyllä tavalla tehtyinä vika- ja vaikutusanalyysit sekä vikapuuanalyysit ovat luotettavuuden arviointitekniikoita, joita sopivasti yhdistämällä aikaansaadaan haluttu tarkastusmenetelmä.

Luotettavuustekniikan termit ovat moniselitteisiä. Vikakäsitteistä käytetään tässä johdonmukaisesti virhettä vikatermin sijasta muistuttamaan siitä, että kaikki ohjelmistoviat ovat suunnitteluvirheitä. Tässä luotettavuudella tarkoitetaan neljää attribuuttia: toimintavarmuus, käyttövarmuus, ylläpidettävyys ja turvallisuus. Luotettavuus-termi vastaa englannin kielen termiä dependability, joka usein vastaa suomenkielessä käsitettä kokonaisluotettavuus ja joskus myös käyttövarmuutta.

2. Tutkimusmenetelmä

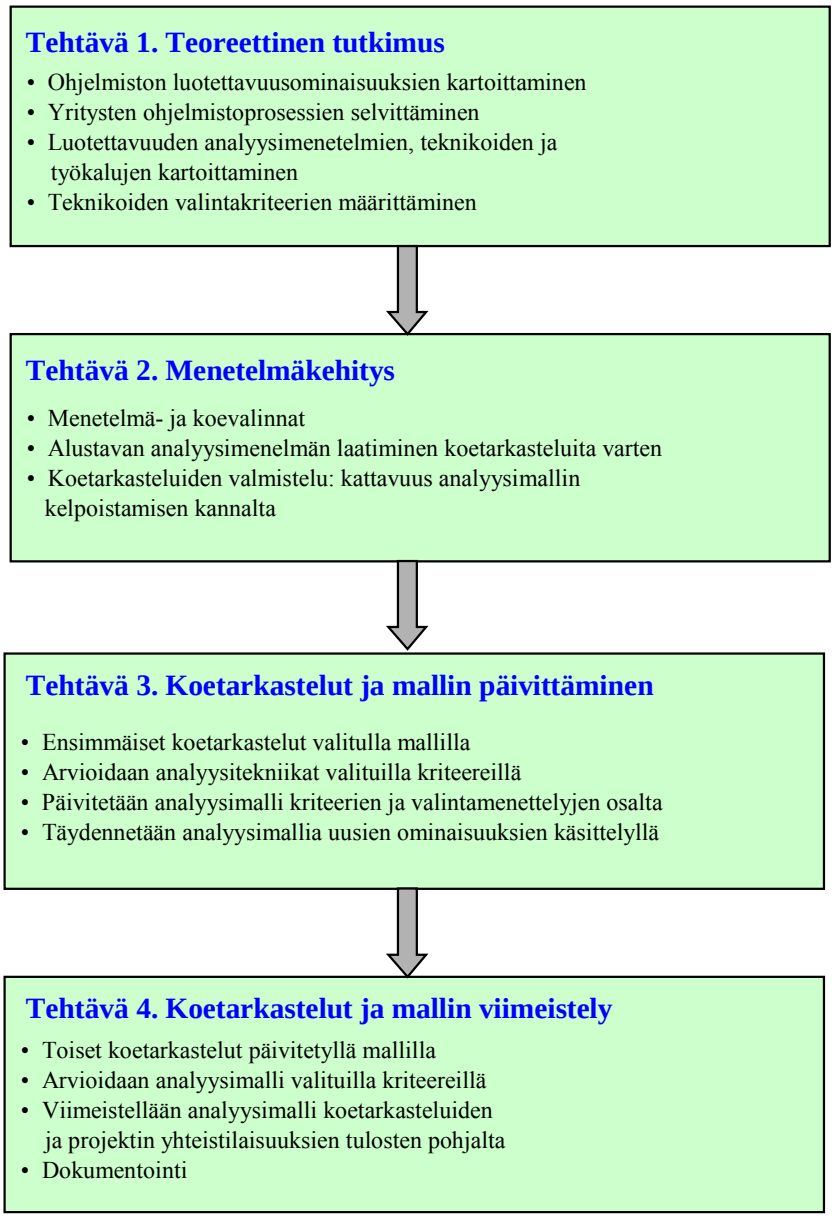
Ohjelmiston luotettavuuden kvalitatiivinen kelpoistusmenetelmä kehitettiin yhdessä prosessiteollisuuden elektroniikan laite- ja järjestelmävalmistajien Honeywell-Measurex Oyj:n ja Neles Automation Oyj:n kanssa kaksivuotisessa projektissa *Kokonaisluotettavuuden integroidun analyysimenetelmän kehittäminen* vuosina 1998–1999. Kokonaisprojekti koostui Tekesin tukemista tutkimusprojekteista ja kummankin yrityksen tuotekehitysprojekteista.

Kehittäminen koostui teoreettisesta mallinkehittelystä ja käytännön koetarkasteluista todellisissa ohjelmistotuotantoympäristöissä. Nämä tehtävät oli jaettu neljään perusvaiheeseen (Kuva 1). Ensimmäisessä vaiheessa keskityttiin teoreettisen tietouden hankintaan luotettavuusominaisuuksista, ohjelmistoprosesseista, analyyseista sekä kriteereistä valita yrityksen ohjelmistotuotantoon sopivat luotettavuuden analyysitekniikat. Valintakriteereinä ovat luotettavuustarkastelun tavoitteet ja tarpeet sekä sen vaatimat resurssit ja sopivuus yrityksen vallitsevaan ohjelmistotuotantoon.

Toisessa vaiheessa laadittiin alustava ohjelmiston luotettavuuden analyysimenetelmä ensi sijassa kolmannessa vaiheessa tapahtuvia yritysprojektien koetarkasteluja varten. Alustava malli koostui muutamasta riskianalyysin perustekniikasta, jotka soveltuvat ohjelmistotuotannon määrittely-, arkkitehtuuri-, suunnittelu- ja toteutusvaiheisiin. Tällaisiksi tekniikoiksi osoittautuvat vika-, vaikutus- ja kriittisyysanalyysi sekä vikapuuanalyysi, jotka ovat koeteltuja perustekniikoita laitteidenkin luotettavuustarkasteluissa. Luotettavuustarkasteluiden, kuten kaikkien riskianalyysienkin, tärkein ominaisuus on systemaattisuus tunnistaa vioittumistapoja, vaikutuksia ja niiden alkulähteitä.

Kolmannen vaiheen koetarkasteluissa tekniikkakohtaista mallia testattiin em. ohjelmiston elinkaarivaiheisiin todellisissa ohjelmistoprojekteissa. Kokemusten pohjalta mallia täydennettiin soveltuvin osin tiettyihin luotettavuusominaisuuksiin kohdistuvilla tekniikoilla, kuten yhteisvikojen käsittelyllä, laitteiston ja ohjelmiston vuorovaikutusten tarkastelulla sekä piilevien polkujen tarkastelulla.

Täydennetty menetelmämalli yhtenä kokonaisuutena koekäytettiin jälleen yritysprojektien koetarkasteluissa projektin viimeisessä eli neljännessä vaiheessa. Tässä vaiheessa selvitettiin myös valitun lähestymistavan merkittävyys ohjelmistopohjaisten järjestelmien luotettavuuden arvioinnissa yritysten kannalta ja ohjaamisessa ohjelmistotuotannon elinkaarivaiheissa. Menetelmämallia sovellettiin ottamalla huomioon sekä yrityksen että ulkopuolisten projektihenkilöiden luotettavuustehtävään soveltuva koulutus.



Kuva 1. Tutkimusprojektin työvaiheet ohjelmiston luotettavuusmenetelmän kehittämisessä. Projektissa oli yhteensä neljä koetarkastelua: Tehtävässä kolme säätöventtiilin digitaalisen asennoittimen ohjelmistokehitys ja kemianlaitoksen turvallisuuteen liittyvän järjestelmän ohjelmistonsovellusprojekti sekä tehtävässä neljä kenttäväylän sovellusprojekti ja paperitehtaan hajautetun automaatiojärjestelmän projekti.

Projektien välisissä yhteistilaisuuksissa vaihdettiin kokemuksia eri valmiusasteisten menetelmämallien hyödyllisyydestä ja koetarkasteluille asetettujen tavoitteiden saavuttamisesta. Aikarajoitusten vuoksi ei tutkimuksessa pyritty kaikenkattavuuteen, vaan tavoitteena oli keskeisimmän mallin kehittäminen ja kuvaaminen. Vaikka kehitetyn mallin tehokkuus onkin sen systemaattisuudessa tunnistaa ja tarkastella virhemahdollisuuksia, ei täydellistä hyvän systemaattisuuden edellyttämää ja erilaisia sovelluskohteita tyydyttävää luokitustapaa ole mahdollista ja ei ehkä tarpeenkaan luoda. Tämäkin malli on ymmärrettävä lähinnä suuntaa antavaksi, ja sen systemaattisia avainlauselistoja pohjaksi yritys- ja projektikohtaisille tarkistuslistoille.

Tutkimus perustui koetarkasteluiden ohessa kirjallisuusselvitykseen. Keskeisimpiä hyödynnettyjä lähteitä ovat olleet ohjelmiston luotettavuusominaisuuksien osalta Laprien teokset [1992, 1998], analyysimenetelmien osalta klassiset jatkuvassa kehityksessä olevat militääristandardit [Def Stan 00-55 1997, Def Stan 00-56 1996, Def Stan 00-58 1996, MIL Stan 882B 1994], jotka ovat myös keskeisiä lähdeteoksia monelle alan tutkimuksista ja menetelmäkehittelystä esittäville teoksille [Leveson 1995, Friedman & Voas 1997, Storey 1996].

Levesonin [1995] teos on jo klassinen oppikirja ohjelmiston vaara-analyyseista kiinnostuneille. Turvallisuuteen liittyvien järjestelmien keskeisin lähde on kuitenkin IEC 61508 [1999]. Näitä asioita Storey [1996] selostaa monen mielestä ymmärrettävämmin. Muita suositeltavia ovat monet instituuttien National Institute of Standards and Technology, Lawrence Livermore National Laboratory sekä Software Engineering Institute lähdeteokset.

3. Ohjelmiston luotettavuusominaisuudet

Ohjelmistovirhe voi syntyä missä vaiheessa elinkaarta tahansa. Sen etenemistä ohjelmiston virhetoiminnoksi ja järjestelmän ulkoisiksi seurauksiksi mallinnetaan kvalitatiivisilla luotettavuustarkasteluilla. Mallintamisessa on tunnettava virheiden synty- ja etenemisehdot eli virhemekanismit, jotta joko virheiltä tai niiden vaikutuksilta vältytään.

Ohjelmaan jäänyt virhe on aina piilevä. Sen olemassaoloa ei tunneta, eikä sitä välttämättä edes ole, mutta juuri siksi sen etsiminen, poistaminen ja kontrollointi kuluttavat rajallisia resursseja. Jotta resurssit pysyisivät hallinnassa on myös syytä asettaa luotettavuustavoitteet – toimintavarmuus, käyttövarmuus, ylläpidettävyys ja turvallisuus – joiden rajaamissa puitteissa ohjelmiston luotettavuus suunnitellaan ja arvioidaan.

3.1 Käsitteet

Laprielle [1985] luotettavuus on järjestelmän keskeinen laatuun liittyvä ominaisuus, jolla perustellaan luottamusta järjestelmän tuottamiin palveluihin eli turvaudutaan järjestelmään. Palvelu on se osa järjestelmän tuottamista tuloksista, jonka käyttäjä havaitsee. Usein luotettavuutta pidetään yleisterminä, jolla tarkoitetaan järjestelmän kaikkia luotettavuusattribuutteja: toimintavarmuutta, käyttövarmuutta, ylläpidettävyyttä ja turvallisuutta. Turvallisuudella tarkoitetaan myös eri asioita. Joskus sen merkitystä laajennetaan käsittämään lähes kaikki, mikä on tavalla tai toisella kriittistä, kuten henkilö-, ympäristö-, omaisuus-, tehtävä- ja käyttökeskeytysturvallisuus.

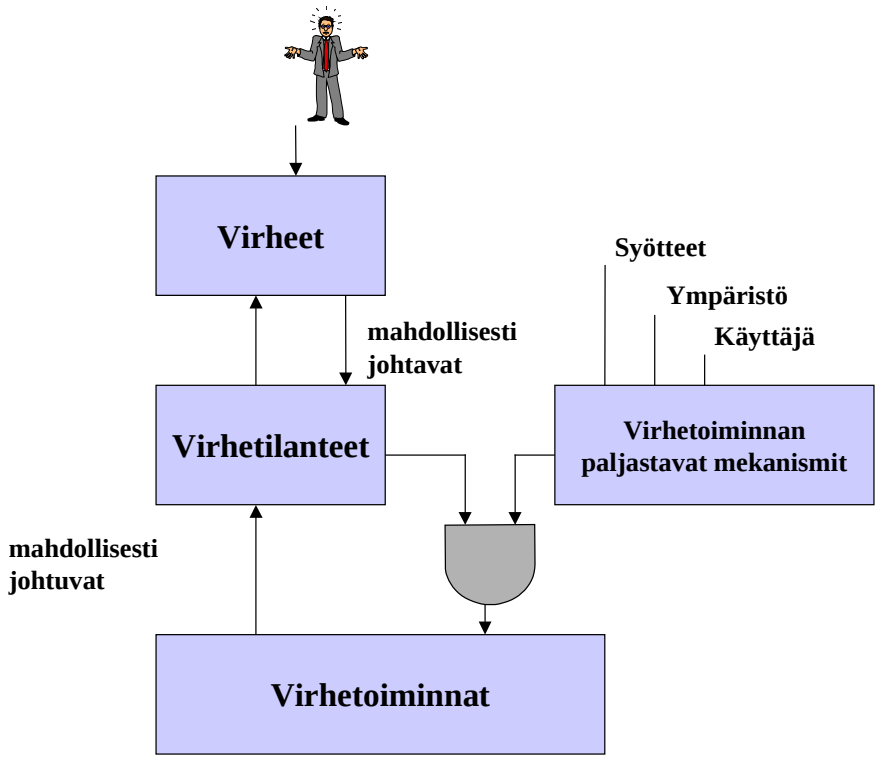
Musa [et al. 1987, 1999] määrittelee ohjelmiston toimintavarmuuden virheettömyytenä tietyissä olosuhteissa ja ympäristössä. Hänen mukaansa ohjelmisto on luotettava spesifikaatioihinsa verrattuna. Palveluun ei olla tyytyväisiä, jos toteutus ei vastaa määrittelyä, ja silloin jossakin kehitysvaiheessa on tehty virhe. Ohjelmiston virheettömyys yhtyy Musan määritelmässä ohjelmiston toimintavarmuuteen. Siinä ei ole kyse toimintojen oikeellisuudesta, ts. ei oteta kantaa siihen, onko ohjelmisto määritelty ja suunniteltu tarkoitusta vastaavasti. Turvallisuuskäsite eroaa toimintavarmuuden käsitteestä tässä suhteessa. Siitä lisää myöhemmin.

Laprie [1985] kokoaa luotettavuuden seuraavista osista:

1. virhemekanismit (heikennykset)
2. keinot (parannukset)
3. luotettavuusattribuutit (kriittisyydet).

Ohjelmiston luotettavuuden heikennyksiä kuvaavat virhekäsitteet virhe (engl. fault), virhetilanne (engl. error) ja virhetoiminta (engl. failure). Virhekäsitteet eivät ole vakiintuneita niin suomenkielisesti kuin englanninkielisestikään. Vikatermiä käytetään yleensä yleismerkityksessä tarkoittamaan lähes kaikkia vikakäsitteitä, virheellä taas viitataan ihmisen tekemään erehdykseen. Viime mainitusta syystä tässä julkaisussa käytetään ohjelmistovioista johdonmukaisesti virhe-termiä.

Virhetoiminta on ohjelman ulkoisesti havaittavan palvelun poikkeama ohjelman spesifikaation mukaisesta toiminnasta eli siitä, mitä järjestelmän pitäisi tehdä. Virhetilanne on se osa järjestelmän tilaa, joka voi mahdollisesti johtaa virhetoimintaan, ja virhe on virhetilanteen todellinen, havaittu tai oletettava syy (Kuva 2).



Kuva 2. Ohjelmiston virhekäsitteet: virhe, virhetilanne ja virhetoiminta. Virhe mahdollisesti johtaa sisäiseen virhetilaan ja ulkoiseen virhetoimintaan. Analysoimalla ja testaamalla selvitetään, mistä virheistä virhetoiminnat ja -tilat johtuvat.

Virhekäsitteet muodostavat jatkuvan ketjun, joka alkaa ihmisen tekemästä virheestä tai ohjelmiston tai laitteiston komponenttitason vikaantumisesta ja jatkuu osajärjestelmän ja järjestelmän vikaantumisiin sekä edelleen järjestelmän ulkoisiin tapahtumiin.

Parannukset luokitellaan virheitä välttäviin ja eliminoiviin, virhesietoisiin ja virheistä ennustaviin menetelmiin. Ne ovat tekniikoita, ratkaisuja ja työkaluja, joita tarvitaan luotettavan tuotteen suunnittelussa, tuottamisessa ja arvioimisessa.

Luotettavuusattribuutit ovat mitattavia ominaisuuksia, joilla tuotteen tai palvelun laadun merkittävyyttä arvioidaan. Tässä tarkastellaan erityisesti neljää attribuuttia: toimintavarmuus, käytettävyys, ylläpidettävyys ja turvallisuus. Kun puhutaan luotettavuudesta, tarkoitetaan aina tietyllä tavalla kriittistä toimintaa.

3.2 Ohjelmiston virhemekanismit

Virhemekanismissa on kyse virhetapahtuman syysuhteesta eli tapahtumavirrasta. Ohjelmistoille syysuhde on kolmidimensioinen käsittäen prosessin, ohjelmiston ja laitteiston. Virheet syntyvät ja etenevät missä tahansa elinkaaren vaiheessa, ohjelmisto- ja laitteistohierarkiassa.

Syysuhteen käsite määritellään eri tavoin riippuen siitä, otetaanko lähtökohdaksi syntaktisuus, semanttisuus vai niiden yhdistelmänä tapahtumafunktionaalisuus. Jos näkökulma on syntaktinen, syysuhde määritellään positionaalisesti, jolloin kausaalinen tapahtumavirta alkaa tietystä muotovirheestä ja kehittyy otollisissa olosuhteissa ohjelmiston sisäisiksi virhetiloiksi ja potentiaalisiksi ulospäin vaikuttaviksi virhetoiminnoiksi. Positio voi sijaita missä vain tapahtumavirrassa. Yleisin syntaktinen virhe on tekstuaalinen koodivirhe.

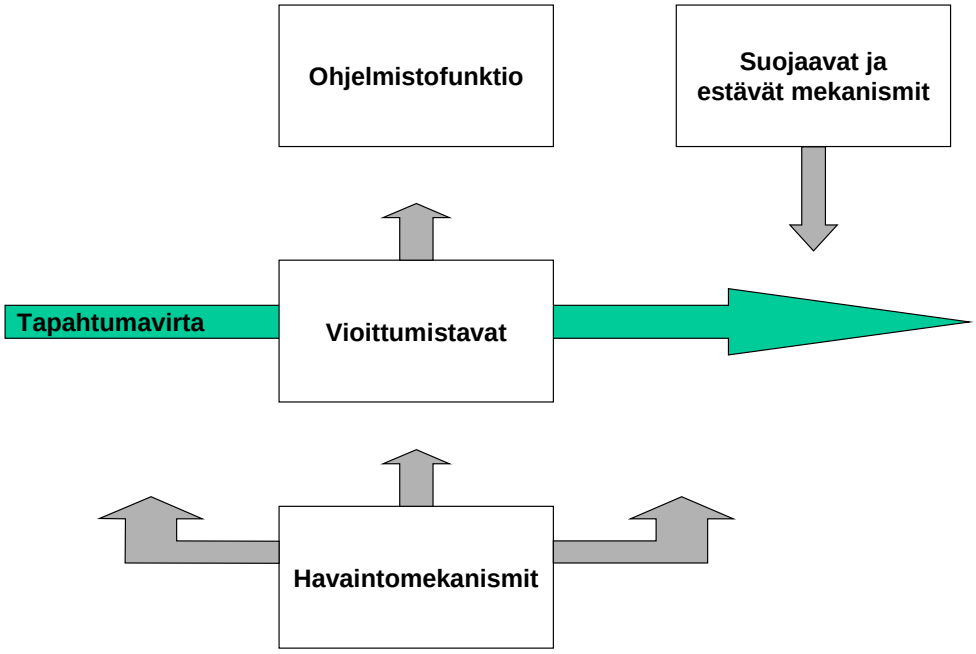
Jos lähdetään semanttisesta eli merkitystehtävästä, syysuhteeksi käsitetään se, mistä tapahtumavirrassa on kyse. Semanttisuus ei riipu positiosta siinä mielessä, että virhelähdettä ei pystytä paikantamaan niin yksikäsitteisesti kuin syntaktista. Virhelähde voi olla missä kohdassa hyvänsä kolmidimensioista tapahtumavirtaa. Esimerkiksi alkuperäinen tavoite on ehkä ymmärretty väärin ja kehitetty virheellinen vaatimus.

Tapahtumafunktionaalinen näkökulma on tietynlainen yhdistelmä edellisistä virhemuodoista. Se on semanttinen siinä mielessä, että tavoitteen merkitystä syntaktisen virheen kautta muunnetaan tapahtumavirrassa. Vaikka syntaktinen lähde eliminoidaan, jää semanttinen virhe jäljelle esimerkiksi vaatimuksen virheellisenä toteutuksena.

Virhemekanismin käsitteet ovat täysin olennaisia yritettäessä purkaa tapahtumasuma, ennen kuin se muodostuu piilevästä näkyväksi ongelmaksi. Käsitellään syysuhdetta vastavirtaan, niin kuin luotettavuustarkasteluissa tehdään, eli ensiksi tunnistetaan mahdolliset ongelmat.

3.2.1 Virhetoiminta

Virhetoiminta on kohteen ulkoisesti havaittavan palvelun poikkeama kohteen spesifikaation mukaisesta toiminnasta eli siitä, mitä järjestelmän pitäisi tehdä [Musa et al. 1987]. Virhetoiminnan seurauksena ohjelman on mahdotonta suorittaa tehtäväänsä. Määritelmä merkitsee myös sitä, että järjestelmällä ei ole virhetoimintaa, jos se toimii spesifikaatioidensa mukaisesti, vaikka jotakin poikkeavaa tapahtuisikin. Tällaisissa tapauksissa kyseessä on virhe spesifikaatiossa, joita luotettavuustarkasteluissa käsitellään erityisesti vaara-analyyseilla.



Kuva 3. Vioittumistapa on keskeinen tarkastelukohde tapahtumavirrassa. Sen syyt ja seuraukset tunnistetaan sekä havainto- ja estomekanismit määritellään.

Eri virhetoiminnoilla on usein erilaisia vaikutuksia järjestelmän toimintaan. Vaikutukset voivat ilmaantua äkillisesti ja odottamatta tai hitaasti niin, että järjestelmän toiminta heikkenee vähitellen. Vain osa toiminnoista on ehkä menetetty, järjestelmä suoriutuu joistakin tehtävistä.

Koska järjestelmät vioittuvat eri tavoin, on johdettu käsite vioittumistapa. Vioittumistapa on keskeinen useimmissa kvalitatiivisissa luotettavuustarkasteluissa. Se on ohjelmiston virhemallissa läheisessä suhteessa virhetoiminnan käsitteen kanssa. Siten voi-

daankin puhua virhetoimintatyypeistä, joille kehitetään luotettavuustarkasteluita varten kuvaavia avainsanoja ja -lauseita.

Vioittumistapa kohdistuu kohteisiin, joiden virhetoiminnoista ollaan kiinnostuneita. Kohteita voivat olla virhetapahtuman merkittävyys, suuruus tai ajastus. Virhetoiminnan syysuhteen syntaktinen, semanttinen tai tapahtumafunktionaalinen luonne selvitetään tunnistamalla syyt ja vaikutukset. Jos selvitys tehdään esimerkiksi järjestelmän keskeydyttyä odottomattomasti, voi syysuhde olla vaikea määrittää. Se riippuu havainnointitavasta, joka ei välttämättä ole johdonmukainen kaikelle järjestelmän käyttäytymiselle.

Vaikutukset luokitellaan järjestelmäympäristössä virhetoiminnan vakavuuden mukaan. Vioittumistavan seurauksen käsite johtaa järjestelmän kriittisyyskäsitteen määrittelyyn, mikä määritellään vakavimman seurauksen mukaan, johon vioittumistavat voivat johtaa.

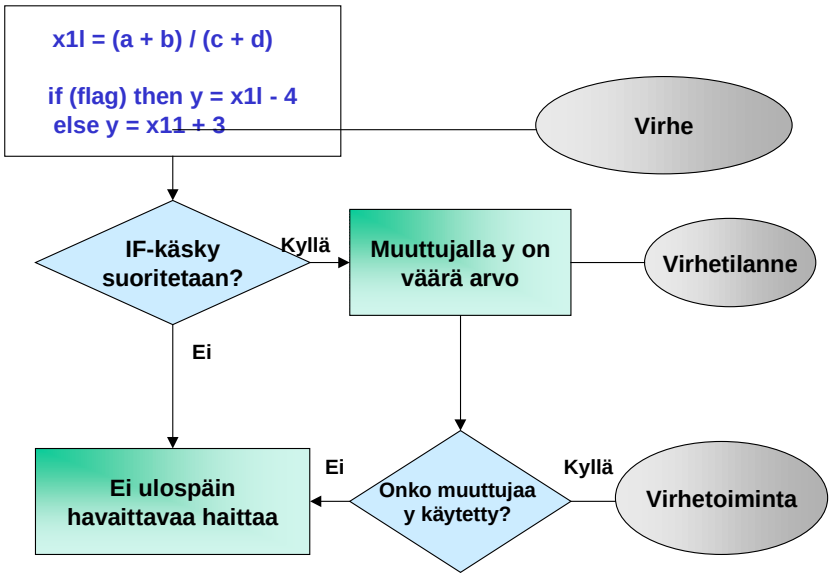
Kuva 3 esittää erästä tulkintaa vioittumistavan ominaisuuksista. Tarkoituksenmukaisuus yleensä ratkaisee syiden ja vaikutusten tunnistamistarkkuuden sekä havainto- ja estomekanismien kohdistuvuuden vioittumistapaan, syyhyn tai vaikutukseen.

3.2.2 Virhetilanteet

Virhetilanne on se osa järjestelmän tilaa, joka voi tietyillä ehdoilla johtaa virhetoimintaan. Erilaisilla virhesietoisilla varmistuksilla joko estetään tietyn virhetilanteen eteneminen tai konstruoidaan yleinen varmistusmekanismi kaikenlaisille mahdollisille virhetilanteille. Kuva 4 esittää yksinkertaista esimerkkiä syntaktisen virheen etenemisestä näkyväksi virhetoiminnaksi. Vain, jos muuttujaa *y* tullaan käyttämään esimerkiksi suorituksessa toiminnallisten testausten yhteydessä, voidaan havaita tietyn tuloksen poikkeavan halutusta tuloksesta ja lähteä jäljittämään virhetilannetta ja poistaa sen syy.

Kuvan tapauksessa virhetoiminta löytyy, kun ohjelma osataan suorittaa tietyillä syötteiden arvoilla ja mahdollisesti vielä tiettynä ajanhetkenä. Virhetilanne on kuvan esittämää tapausta paljon monimutkaisempi silloin, kun kyse on semanttisesta syysuhteesta tai ilkivaltaisesta ohjelmoinnista. Semanttisessa koodinosan merkitys on saattanut muuttua esimerkiksi virheellisen korjauksen jälkeen niin, että vaikka tiedetäänkin virhetoiminnon läsnäolo, sitä ei osata jäljittää oikeaan kohtaan, vaan mahdollisesti tehdään virheellinen muutos toiseen osaan ohjelmaa.

Ilkivaltaisessa ohjelmoinnissa on tahallisesti koodattu ohjelmanosia ja niitä herättäviä syötemahdollisuuksia. Näitä ovat salaovet ja takaportit, joita ei spesifioida vaatimuksiksi näkyvästi vaan peitellysti siten, että testaaminen vaikeutuu.



Kuva 4. Virhetilanne muuttujan y käytössä. Jos suorituspolku kulkee else-lauseen kautta, syntaktinen kirjoitusvirhe on ohjelman virhetilanne. Jos muuttujaa y hyödynnetään muualla ohjelmassa, virhetilanne etenee ohjelmiston ulospäin näkyväksi virhetoiminnaksi.

3.2.3 Virheet

Virhe on virhetilanteen havaittu tai oletettu syy. Se voi olla fyysinen tai inhimillinen, tahallinen tai tapaturmainen. Se voi syntyä missä prosessointivaiheessa tahansa, jolloin on vastaavasti kyse määrittely-, suunnittelu- tai käyttövirheistä. Virhe voi olla ohjelmiston sisäinen tai ulkoinen laitteisto-, järjestelmä- tai ympäristövirhe. Virhe on aina jossakin muodossa syntaktisena tai semanttisena tarkasteltavassa ohjelmassa tai sen laitteistossa, mutta se on prosessin tuote.

Pressman [1997] jakaa ihmisen tekemät ohjelmistovirheet seuraavasti:

1. Suunnitteluvirheet ovat kehitystyön aikaisia, tahattomia tai tapaturmaisia ilman vahingoittamisen tarkoitusta. Vaatimukset on väärin tulkittu ja siten toteutettu. Ohjelmiston suunnitteluvirheet ovat aina eliminoitavissa ja korjattavissa uudelleensuunnittelulla.
2. Vuorovaikutteiset virheet ovat ulkoisia virheitä ilman tahallisen vahingoittamisen tarkoitusta.

3. Ilkivaltaisia sisäisiä virheitä ovat mm. virukset, madot, Troijan hevoset, salaovet, loogiset- ja aikapommit.
4. Tunkeutumiset ovat ilkivaltaisia ulkoisia virheitä. Ne ovat mahdollisia vain, jos järjestelmässä on jokin määrätty suunnitteluvirhe.

Muutokset ja korjaukset ovat virhetilanteeseen johtavista syistä eräitä yleisimpiä. Ulkoisen ympäristön vaatimukset voivat muuttua, suunnittelussa on saatettu havaita muutos- tai parantamistarvetta, jota ei ole viety laadunvarmistuksen prosesseihin.

Virheet voidaan luokitella myös niiden säilyvyyden mukaan. Leveson [1986] erottaa kolmentyyppisiä virheitä: tilapäiset, pysyvät ja ajoittaiset, jotka voivat olla piileviä tai havaittavia. Yleensä kaikki virheet ovat piilevinä ainakin jonkin aikaa, kunnes ne tietyillä mekanismeilla havaitaan. Tilapäiset virheet ilmestyvät ja vaikuttavat hetken ja häviävät. Ne aiheuttavat tietokonejärjestelmälle virhetoiminnan, joka on poistunut järjestelmää uudelleenkäynnistettäessä. Niiden syyt ovat tietysti vaikeasti selvitettäviä, usein staattisen sähkön aiheuttamia.

Pysyvät virheet ilmestyvät tiettynä kehitysprosessin hetkenä ja jäävät määräämättömäksi ajaksi. Ne ovat piileviä virheitä, jotka tietyillä ehdoilla etenevät virhetilaan. Ohjelmiston suunnitteluvirheet ovat pysyviä niin kauan, kun ne korjataan uudelleensuunnittelulla. Yksi suunnitteluvirhe voi aiheuttaa useita virhetilanteita ja virhetoimintoja, ennen kuin se riittävän kattavan diagnoosin kautta kokonaan korjataan. Ajoittaiset virheet ilmestyvät ja häviävät, usein johtuen lämpöherkkyydestä.

Hecht & Hecht [1986] osoittavat tutkimuksissaan ohjelmistovirheen esiintyvän suunnilleen joka 50. ohjelmarivillä. 90 % näistä virheistä löydetään testeillä, ja jäljelle jääneistä valtaosa löydetään ensimmäisen käyttövuoden aikana. Ylläpitotoimet poistavat osan jäljelle jääneistä virheistä, mutta tutkijat väittävät samalla ilmestyvän uusia virheitä lähes yhtä paljon.

Beizer [1990] on luokitellut ohjelmistovirheet niiden syntytavan mukaan. Hänen luokittelussaan on useita tasoja, joista toiminnallisten vaatimusten ja ominaisuuksien spesifioinnissa ja toteuttamisessa voi esiintyä seuraavanlaisia virhetyyppejä:

1. Vaatimus on väärä tai virheellinen, ei-toivottu. Vaatimus voi olla oikein määritelty mutta ei haluttu, tarpeeton tai ylimääräinen.
2. Vaatimus on epälooginen: ristiriitainen ja havaitaan yleensä staattisissa analyyseissa. Tai se voi olla
   - kohtuuton: looginen ja johdonmukainen, mutta rajoitteisiin sopimaton

- saavuttamaton: mahdoton vaatimus esimerkiksi toteutettavaksi annetuilla resursseilla
- yhteensopimaton: ei sovi muiden vaatimusten tai ympäristön yhteyteen
- sisäinen: ilmeinen virhe tietyssä komponentissa
- ulkoinen: ristiriitainen muiden komponenttien kanssa
- yhteensopimaton: vaatimus on yhteensopimaton laitteiston, ohjelmiston tai käyttöjärjestelmän kanssa.

3. Vaatimus on vaillinainen: Vaillinainen määrittely – variaatiot, attribuutit, ominaisuudet ym. ovat määrittelemättä. Tai vaatimus voi olla
   - puuttuva: vaatimusta ei ole määritelty
   - päällekkäinen: vaatimus on määritelty jo muualla
   - geneerinen: vaatimus on oikea ja ristiriidaton mutta liian yleinen sovellettavaksi.
4. Vaatimus ei ole todennettavissa: annetuilla resursseilla vaatimus on mahdoton näyttää toteen millään keinolla. Esimerkiksi oikeat testit voidaan suunnitella mutta ei toteuttaa, tai vaatimukseen liittyy
   - puutteellinen dokumentaatio: vaatimukset ovat oikeita, mutta esitysmuoto ei
   - virheet standardeissa: vaatimusstandardit, joiden perusteella vaatimus on määritelty, sisältävät virheellistä tietoa.

3.3 Luotettavuutta kasvattavat keinot

Luotettavuutta kasvattavat keinot ovat suunnittelussa ja toteuttamisessa tarvittavia menetelmiä, tekniikoita, työkaluja ja ratkaisuja. Niillä kyetään sekä toimittamaan luotettava tuote että vakuuttautumaan luotettavuuden riittävyydestä. Ne voidaan luokitella usealla tavalla, Laprie [1998] jakaa ne kolmeen ryhmään virheitä 1) välttäviin ja poistaviin, 2) sietäviin ja 3) ennustaviin. Virheitä välttävät keinot tukevat virheetöntä suunnittelua, virheiden poistamisen tärkeimmät keinot ovat staattiset analyysit ja testit, virhesietoiset keinot ylläpitävät toimintaa virhetilanteista huolimatta ja virheitä ennustavat keinot estimoivat nykyisten ja tulevien virheiden lukumäärää. Tarkastellaan näiden keinojen merkitystä luotettavuuden tarkastelun ja ohjaamisen kannalta. Varsinaiset ohjelmiston luotettavuuden analyysitekniikat esitetään luvussa viisi.

3.3.1 Virheiden välttäminen ja poistaminen

Virhetilanteen syntymistä lopputuotteeseen ehkäistään joko välttämällä virheiden syntymistä tai tunnistamalla, diagnosoimalla ja korjaamalla ne. Pressman [1997] esittää, että mahdollisimman oikean ohjelmiston tuottaminen edellyttää 1) tarkkaa spesifiointia, 2) todistettavasti hyviksi todettujen suunnittelumenetelmien käyttöä, 3) abstraktien tietotyyppien ja modulaarisuuden käyttöä sekä 4) tukiympäristön käyttöä.

Turvallisuuteen liittyvä standardi IEC 61508 [1999] suosittaa virheiden välttämiseksi seuraavia suunnittelu- ja toteutusmenetelmiä:

- rakenteellinen suunnittelu siten, että systeemi- ja tietorakenteiden, ajan- ja resurssienkäytön, liitäntöjen sekä muiden toimintojen suunnittelu ennen yksityiskohtia
- rakenteellinen toteuttaminen
- modularisointi siten, että moduulin kokoa ja monimutkaisuutta rajoitetaan
- strukturoitu ohjelmointi siten, että ohjelmarakenne rajoittuu sekvenssiin, silmukkaan, valintaan ja alirutiinikutsuun
- kaaviot pidetään minimaalisen kokoisina
- koodaus-, nimeämis- ja dokumentointisäännöt.

Virheen poistaminen riippuu virheen tunnistamistavoista kehitysprosessissa. Tapoja ovat mm. suunnittelukatselmukset, ohjelman todentaminen, koodintarkastus ja järjestelmätestit. Järjestelmätestit ovat tärkein tunnistamistapa, mutta testeilläkin on omat ongelmansa:

- Testaus ei voi koskaan olla täysin kattavaa etenkään laajoissa systeemeissä.
- Testauksella osoitetaan virheiden olemassaolo, mutta ei niiden poissaoloa.
- Testaus todellisin ehdoin (oikeassa ympäristössä) voi olla mahdotonta.
- Spesifiointivirheet voivat näkyä vasta loppukäytössä.

Koska monimutkaiset ja laajat ohjelmistojärjestelmät sisältävät joka tapauksessa virheitä ja niiden esiintymistä ei voida täysin välttää eikä niitä voida täysin poistaa, tarvitaan virhesietoista suunnittelutapaa, josta lähemmin kohdassa 3.3.2 Virhesietoisuus.

Todennusprosessissa tarkastetaan vaiheittain, täyttääkö järjestelmä sille asetetut ominaisuudet eli todennusehdot. Todennusehdot ovat joko suhteellisen riippumattomia spesifikaatiosta, yleisiä tai hyvin spesifistisiä tietyille systeemeille. Todennus voi olla staattista tai dynaamista.

Todennus on staattista toimintaa silloin kun ohjelmaa tai järjestelmää ei suoriteta. Staattisia analysointeja ovat todennettavaan järjestelmään pohjautuvat erilaiset tarkistukset: tietovuoanalyysi, kompleksisuusanalyysi, ohjelmakääntäjän tarkistukset, turvallisuusanalyysit sekä analysointitapaa läheisesti muistuttavat matemaattiset todistukset. Todennus voi myös perustua järjestelmää kuvaavaan malliin.

Todennus on dynaamista silloin, kun ohjelma tai järjestelmä pitää sitä varten suorittaa. Testaaminen on yksi dynaamisen todentamisen muoto. Niitä on erilaisia ja luokittelutapojakin on useita kymmeniä. Testeillä tavoitellaan joko spesifikaationmukaisuutta tai virheiden etsintää. Testit voivat perustua järjestelmämalleihin, jolloin ne ovat joko toiminnallisia tai rakenteellisia tai spesifistisiä tiettyihin vioittumistapoihin perustuvia malleja.

Testisyötteiden generoinnissa ovat erotettavissa deterministinen ja tilastollinen tapa. Edellisessä testitapaukset valitaan määrättyjen kriteerien mukaan, jälkimmäisessä syötekentän todennäköisyysjakauman pohjalta.

Käytön aikainen virheenpoisto on korjaavaa ylläpitoa, joka voi olla parantavaa tai ennakoivaa. Parantavassa raportoidut virheet korjataan, ennakoivassa virheet pyritään korjaamaan, ennen kuin ne näkyvät järjestelmän toiminnassa. Viimeksi mainittuja virhetyyppejä ovat fysikaaliset virheet, jotka on havaittu edellisen ennakoivan ylläpidon jälkeen, sekä suunnitteluvirheet, jotka on havaittu vastaavissa muissa systeemeissä.

3.3.2 Virhesietoisuus

Ohjelmistovirhe syntyy aina inhimillisten virhetekijöiden vaikutuksesta, määrittely-, suunnittelu- tai koodausvirheenä. Virheitä ja niiden vaikutuksia pyritään eliminoimaan paitsi kehityksenaikaisilla tarkistuksilla ja testauksilla myös rakentamalla laitteisto ja ohjelmisto virhesietoiseksi. Virhesietoisuudella tarkoitetaan järjestelmän kykyä toimia virheettömästi virheiden läsnä ollessa. Virhesietoinen ohjelmisto toimii oikein, vaikka sen koodissa olisi virheitä.

Virheitten ilmaannuttua on useita vaihtoehtoisia virhesietoisia toimintatapoja:

- Täydellinen virhesietoisuus – järjestelmä toimii virheettömästi jonkin aika, kunnes virheitä kertyy liiaksi.
- Hallittu hajoaminen – järjestelmän tärkeitä toimintoja ylläpidetään toissijaisten kustannuksella, ennen kuin toipuminen tai korjaus on tehty.
- Vikaturvallisuus – järjestelmä johdetaan turvalliseen tilaan joko pidemmäksi aikaa tai mahdollisesti tilapäisesti siksi, kunnes virhe on korjattu.

Virhesietoiset menettelytavat määritellään sen mukaisesti, mihin virheen etenemisvaiheeseen niillä pyritään vaikuttamaan. Virhetilanteen käsittelyllä pyritään katkaisemaan virheiden kulkeutuminen virhetoiminnaksi, virheen käsittelyllä pyritään estämään virheitä aktivoitumasta virhetilaksi ja siten edelleen virhetoiminnaksi. Virhesietoiset menettelytavat vaihtelevat myös virhetyypin mukaan. Seuraavaksi käsitellään virhesietoisuutta järjestelmän (lähinnä fyysisten virheiden kannalta) ja suunnittelun osalta. Laprie [1998] tarkastelee myös käyttövirheisiin kohdistuvia virhesietoisuusmenetelmiä, mutta tässä yhteydessä ne jätetään käsittelemättä.

Järjestelmän virhesietoisuus on virhetilanteen käsittelyä, joka jaetaan kolmeen vaiheeseen: 1) virhetilanteen ilmaisu, 2) virhetilanteen diagnosointi ja 3) virhetilanteesta toipuminen. Virheen havaitsemisella tarkoitetaan yleensä virhetilanteen ilmaisua. *Virhetilanteen ilmaiseminen* on virhesietoisen järjestelmän perusedellytys. Sillä pyritään havaitsemaan [Bjarland 1986]

- poikkeustilanteita – tilanteita, joissa järjestelmän jonkin osan tai komponentin vaste on ei-toivottu, virheellinen tai epänormaali
- tiedon, syötteiden ja herätteiden arvoja ja arvojen yhdistelmiä, jotka saattaisivat virheen aktivoimisella aiheuttaa virhetilanteen.

Fyysisiä virheitä vastaan on kehitetty lukuisa joukko tarkastustekniikoita. Periaatteena on luoda esteitä, joilla torjua virheen aktivoituminen ja eteneminen virhetoiminnaksi. Ne luokitellaan seuraavasti:

1. Virhetilanteen ilmaisumenetelmät. Koodattuun tietoon on lisätty redundanteja koodeja, esimerkiksi pariteettibitti.
2. Redundanttinen tarkistustekniikka. Kahdennettuja kohteita tai niiden toimintoja verrataan keskenään. Jos verrattavia kohteita on enemmän kuin kaksi, kyse on äänestyksestä.
3. Käänteistarkastus. Tuloksesta käänteisellä algoritmilla lasketaan syötteille arvot, joita verrataan käytettyihin arvoihin.
4. Ajastus- ja suoritustarkistukset. Ne toteutetaan esimerkiksi vahtikoiralla.
5. Resurssien käytön tarkistukset.
6. Järkevyystarkastukset. Niitä ovat arvoalueen tai tyypin, arvojen välisen ristiriidattomuuden, arvon muutosnopeuden, indeksin ja osoittimien rajojen sekä ohjauksen siirron tarkistukset.
7. Rakenteelliset tarkastukset.

Virhetilanteen diagnosointi voi olla automaattista, itsediagnostiikkaa, joka suoritetaan joko laitteistolla tai ohjelmistolla. Virhetilanteesta toipumisen tavoitteena on eliminoida tai kompensoida virhetilanteesta syntynyt vahinko niin, ettei se johda virhetoimintaan.

Yleensä *virhetilanteesta toipuminen* on joko takautuvaa toipumista, jolloin järjestelmä tuodaan virheellisestä tilasta aikaisempaan tiettyyn virheettömään tilaan, tai ennakoivaa toipumista, jolloin vastaavasti järjestelmälle löydetään uusi tietty toimiva käyttötila. Lisäksi toipuminen voi tapahtua kompensoinnilla siten, että järjestelmän siirto virheettömään tilaan onnistuu. Takautuva toipuminen suunnitellaan mielivaltaisia vioittumistapoja vastaan, sen sijaan ennakoiva toipuminen sopii vain tietyille, yksittäisille vioittumistavoille, jotka siis voidaan määritellä. Viimemainittu ominaisuus on ohjelmistovirheiden kohdalla hankalasti suunniteltavissa, joten takautuva toipuminen on ohjelmistoille merkittävämpää. Ennakoivan toipumisen mekanismi ei kykene tehokkaasti käsittelemään ohjelmistovirheistä aiheutuvia virhetilanteita. Takautuva toipuminen vaatii käsittelyohjelmalta niin suurta tilantarvetta, että suorittaminen hidastuu.

Virheen käsittelyyn liittyvät vastaavat toiminnot kuin virhetilanteen käsittelyyn: virheen diagnosointi, virheen eristäminen ja toipuminen uudelleenkonfiguroinnilla. Virheet, joita on pakko sietää, luokitellaan ennustettaviin ja ennustamattomiin virheisiin, joista jo edellä oli puhetta. Ennustettavien virheiden eteen joudutaan työskentelemään enemmän, koska niiden ominaisuuksista tiedetään enemmän. Vastaavasti tiedetään myös niiden esiintymistiheydestä ja vuorovaikutuksesta järjestelmän kanssa. Vuorovaikutukseen liittyvät laitteiston virhetoiminnat, liittymät, odotettavissa oleva käyttö ja siihen liittyvä problematiikka, ulkoisen järjestelmän virhetoiminnat, jne.

Ennustamattomiin virheisiin sisältyvät jotkut suunnitteluvirheet ja sellaiset ongelmat, jotka ilmenevät vasta todellisessa käyttöympäristössä järjestelmän käyttöönoton jälkeen. Ennustamattomille virheille on tietysti vaikeaa rakentaa vikasietoisuutta juuri niiden luonteesta johtuen.

Kaikki vikasietoisuustekniikat riippuvat aina jollakin tapaa redundanssista. Redundanssia voivat olla myös ohjeet ja järjestelmän komponentit, joita ei normaalitoiminnassa tarvita.

Suunnittelun virhesietoisuudessa pitää muistaa, että ohjelmistovirhe on tyypillinen systemaattinen suunnitteluvirhe. Ohjelman kopiointi merkitsee myös virheen kopioimista, sama virhe on kaikissa rinnan ajettavissa samoissa ohjelmissa. Moninkertaistamisesta ei ole siten hyötyä. Ohjelmiston, kuten yleensäkin suunnittelun, virhesietoisuudelle on löydettävä muita keinoja. Niitä löytyy kahdenlaisia:

1. Suunnittelun erilaisuus perustuu siihen, että vähintään yksi ylimääräinen osa tai komponentti suorittaa saman tehtävän erillisellä suunnittelulla ja toteutuksella samasta spesifikaatiosta.
2. Suunnitellaan virhealttiit kohdat siten, että virhetilanne ei johda ohjelmiston virhetoiminnoksi.

Suunnittelun erilaisuudessa eli diversiteetissä on vähintään kahden erillisen komponentin lisäksi oltava päätöksentekoa varten moduuli, jolla ilmaistaan virheetön tulos erillisten komponenttien antamien tulosten perusteella. Spesifikaatiossa tulee ilmoittaa sekä päätöksenteon suorittamisen kohdat että sen käyttämät tiedot. Yhteinen spesifikaatio on yksi suunnittelun diversiteetin heikkouksia.

Suunnittelun erilaisuusperiaatetta hyödynnetään lähinnä kriittisissä henkilöturvallisuuteen liittyvissä systeemeissä joko vikaturvallisesti tai turvatoiminnon jatkuvuuden varmistamiseksi. Käyttö vaihtelee myös kohdealueittain, mutta painottuu ilma- ja rautatieliikenteeseen. Teollisuusprosessien kohdealue on vähäisempää. Virhesietoisuustekniikoista voisi mainita toipumislohkot, N-versio- ja N-itsetarkistusohjelmoinnin.

Samansukuisten virheiden välttämisessä on tärkeää eri versioiden tai moduulien riippumattomuus. Mahdollisimman suuren riippumattomuuden saavuttamiseksi tulisi eri versioiden ja moduulien kehitystyössä käyttää eri algoritmeja, kieliä, kääntäjiä, työkaluja, toteutustapoja, testimenetelmiä, jne. Myös ohjelmistosuunnittelijoiden ja ohjelmoijien erilainen kokemus- ja koulutustausta sekä keskinäisen kommunikoinnin välttäminen ohjelmiston kehitystyöasioissa on tärkeää.

3.3.3 Virheiden ennustaminen

Virheiden ennustamisessa arvioidaan järjestelmän käyttäytymistä virheiden esiintymisen ja aktivoitumisen yhteydessä. Arvioinnissa tarkastellaan toimintojen ja komponenttien virhetaajuuksia ja virheiden vaikutuksia. Tarkastelutapoja on kaksi:

1. Kvalitatiivinen merkittävyyden arviointi, missä tunnistetaan ja luokitellaan virhetoimintoja sekä määrätään sopivia menetelmiä ja tekniikoita mahdollisilta virhetoiminnoilta välttymiseksi.

2. Kvantitatiivinen merkittävyyden arviointi, missä todennäköisyyspohjaisin keinoin arvioidaan tiettyjen luotettavuusattribuuttien merkittävyysastetta.

Menetelmät ja tekniikat ovat joko spesifistisiä kvalitatiivisille arvioinneille (mm. vika- ja vaikutusanalyysi) ja kvantitatiivisille arvioinneille (Markovin analyysi) tai hyödynnettävissä kumpaankin arviointitapaan (vikapuuanalyysi). Tämä julkaisu keskittyy kvalitatiivisiin menetelmiin, joita tarkastellaan lähemmin luvussa viisi.

Kvantitatiivisesti ohjelmiston luotettavuus voidaan määritellä todennäköisyytenä, että tarkasteltava ohjelma ei aiheuta virhetoimintoa määrätyissä olosuhteissa määrätyllä aikavälillä. Pyritään numeeriseen arvioon hyödyntämällä luotettavuusmalleja. Luotettavuusmalleja on kehitetty iso joukko. Singpurwalla & Wilson [1994] ovat katsastaneet edustavan joukon ennustemalleja. Mallit perustuvat joko graafisiin tai analyyttisiin kuvauksiin systeemistä. Graafisissa kuvauksissa luotettavuusarvio aikaansaadaan allokoimalla mallin parametreihin satunnaiset todennäköisyysarvot. Yleisemmin käytettyjä graafisia kuvauksia ovat lohkokaaviot, vikapuut, Markovin kaaviot ja stokastiset Petriverkot. Näistä vikapuuta voidaan soveltaa myös pelkästään kvalitatiivisesti. Graafiset mallit voivat olla myös ohjelman sisäisiä, analyyttiset käsittelevät ohjelmaa lähes yksinomaan mustana laatikkona.

Analyyttisissä luotettavuusmalleissa ohjelmistoa tarkastellaan kokonaisuutena; vain sen liitynnät ympäristöön mallinnetaan. Malleja sovelletaan elinkaaren myöhemmissä vaiheissa, joten niistä ei ole apua ohjelmistoprosessien ohjauksessa. Yksinkertaisimmillaan ja yleisimmin ohjelmiston luotettavuus lasketaan staattisesti onnistuneiden testisyötteiden suhteena syötteiden kokonaismäärään. Pelkästään virheiden lukumäärä ei ole kuitenkaan verrannollinen luotettavuuteen, jossa aina on kyse jossakin määrissä kriittisyydestä. Staattisen mallin heikkous on myös siinä, että tuskin koskaan voidaan testata koko syöteavaruutta.

Oletetaan, että ohjelmisto suoritetaan tietyillä syötteillä, joiden lukumäärä on $N_0(t)$, missä t kuvaa syötteiden suoritusaikaa. Olkoot $N_s(t)$ hyväksyttyihin suorituksiin kuuluvien syötteiden lukumäärä ja $N_f(t)$ niiden syötteiden lukumäärä, jotka aiheuttivat ohjelmistossa virhetoiminnon siten, että $N_s(t) + N_f(t) = N_0(t)$. Ohjelmiston toimintavarmuudeksi $R_f(t)$ saadaan siten virhetaajuutena

$$R_f(t) = 1 - N_f(t)/ N_0(t). \quad (1)$$

Jotta $R_f(t)$ olisi todellinen ohjelmiston luotettavuus tietyllä ajanhetkellä t, tulisi suorittaa kaikki mahdolliset syöteavaruuden arvot. Käytännössä tämä usein merkitsee ääretöntä määrää syötteitä. Mallia voidaan tarkentaa olettamalla syöteavaruudesta esimerkiksi

asiantuntija-arvioilla rajoittavia tekijöitä. Dynaamisten mallien perusajatus on se, että kun ajassa etenevän testauksen yhteydessä havaitaan virheitä, ne poistetaan, jolloin uusien virheiden paljastumistaajuuskin pienenee. Ohjelmiston luotettavuus paranee iän myötä, jos uudistuksia ei tehdä liiaksi.

3.4 Luotettavuusattribuutit

Attribuutit ovat mitattavia ominaisuuksia, joilla tuotteen tai palvelun laadun merkittävyyttä arvioidaan. Tässä yhteydessä tarkastellaan neljää luotettavuusattribuuttia: toimintavarmuus, käyttövarmuus, ylläpidettävyys ja turvallisuus. Ne edustavat ohjelmiston ja järjestelmän perusominaisuuksia. Lisäksi etenkin reaaliaikaisille systeemeille tunnistetaan muitakin attribuutteja, kuten tietoturva, käyttökelpoisuus ja ajastus.

Ohjelmiston laatu esitetään lukuisilla tavoilla. Luotettavuus sisältyy kaikkiin esitystapoihin, olivat kuvauskäsitteenä ominaisuudet, kriteerit tai attribuutit. Koska luotettavuus kuuluu laatuattribuutteihin, ohjelmiston luotettavuus riippuu ohjelmiston laadusta. Laatu rakennetaan tuotteen elinkaaren aikana kaikissa vaiheissa, mistä seuraa, että luotettavuus riippuu myös elinkaaren vaiheista. Erityisesti kustannustehokkaan luotettavuuden rakentaminen aloitetaan mahdollisimman varhaisista elinkaaren vaiheista tunnistamalla virheherkät alueet sekä ehkäisemällä niiden syntymistä.

3.4.1 Toimintavarmuus ja käytettävyys

Toimintavarmuus on järjestelmän tuottamien palveluiden jatkuvuutta, ts. ohjelmiston kyky säilyttää toiminnan taso tietyissä olosuhteissa ennalta määritellyn ajan. Käyttäjä odottaa järjestelmän toimivan tietyn ajan. Koska toimintahäiriöt voivat aiheutua mistä ohjelmiston kehitysvaiheessa tehdystä virheestä tahansa, myös toimintavarmuusattribuutti kuuluu kaikkiin ohjelmistotuotannon vaiheisiin. Usein toimintavarmuudella ymmärretään myös käsitteitä tarkkuus, ristiriidattomuus, robustisuus (kestävyys) tai kyky toimia epänormaaleissa olosuhteissa. Luotettavuudelle on kehitetty useita mittaustapoja (mm. *MTTF*, vikatiheys) ja ennustemalleja (mm. luotettavuuden kasvumallit).

Toimintavarmuus liittyy järjestelmän ennakoitavaan virhetoimintaan, käytettävyys taas palvelun käyttövalmiuteen. Käyttövalmiuteen kuuluvat helppous käyttää ja oppia käyttämään ohjelmistoa mm. vaadittavien syötteiden muodostamisessa sekä tulosten tulkitsemisessa. Käytettävyyttä tarvitsevat kaikki järjestelmät jossakin määrin, toimintavarmuutta, ylläpidettävyyttä ja turvallisuutta vain tietyt järjestelmät. Käytettävyys mitataan todennäköisyyden raja-arvona tietyllä ajanhetkellä t, kun t lähestyy ääretöntä. Kyse on silloin järjestelmän jatkuvuustilan käytettävyydestä, mikä voidaan laskea seuraavasta yhtälöstä:

$$Käytettävyys = MTTF / (MTTF + MTTR), \quad (2)$$

missä *MTTF* on keskimääräinen vikaantumisaika ja *MTTR* keskimääräinen korjausaika.

Pitkäaikaiselle toimintavarmuudelle ei yleensä järjestelmästä riippuen aseteta korkeita vaatimuksia. Esimerkiksi tyypillisen turvalliseen tilaan ajavan suojausjärjestelmän käytettävyys saattaa olla hyvin korkea, mutta vaatimus toimintavarmuudelle matala johtuen siitä, että järjestelmän ei tarvitse olla toiminnassa kuin lyhyen ajan. Turvallisuuden eheyden käsite on kuitenkin hieman monimutkaisempi, sillä siinä edellytetään suojausjärjestelmän olevan tiettyä turvallisuuden eheystasoa myös toimintaa odottavassa tilassa. Turvallisuuden eheys onkin siten verrattavissa korkeaan toimintavarmuuteen.

Toimintavarmuuden ja käytettävyyden mukaanotto ohjelmistokehityksen prosesseihin tulisi ottaa huomioon jo mahdollisimman varhaisessa vaiheessa. Tulee ennakoida, mitkä suunnitteluratkaisut ovat virhealttiita. Ohjelmiston määrittelyvaiheessa (tai järjestelmän tai käyttäjän määrittelyssä) kaikki projektin ohjelmistot luokitellaan tiettyyn vakavuustasoon. Vakavuustasoihin määrääminen on alustava kriittisyyden luokittelutapa, josta luokittelua täsmennetään ohjelmistokehityksen kuluessa ohjelmiston toimintoihin, osatoimintoihin, komponentteihin, jne. Tiettyihin, ennalta laadittuihin vakavuustasoihin määrittely ei yleensä ole hankalaa, mutta jos halutaan täsmällistä määrittelytapaa, voidaan käyttää sopivaa vioittumistapa-, vaikutus- ja krittiisyysanalyysia (esim. FMECA:ta). Vaikutukset ja kriittisyydet voidaan saada selville järjestelmä- tai käyttäjämäärittelyn tasolla, tarkemmin kuitenkin ohjelmiston määrittelydokumentaatiosta. Ohjelmistomäärittelyssä tunnistetaan kriittisiin järjestelmävikaantumisiin johtavat ohjelmistotoiminnot ja määritellään asianomaiset toimenpiteet (suojaukset ja muut uudet vaatimukset, lisäanalysoinnit tai testien painottamiset).

Toimintavarmuuden käsite on matemaattisesti selkeä, mutta kvalitatiivisesti sillä on useita tulkintoja. ISO/IEC 9126 [1991] tulkitsee sen kypsyytenä, virhesietoisuutena ja toipumisena, McCabe [1976] tarkkuutena, virhesietoisuutena, johdonmukaisuutena ja yksinkertaisuutena sekä Boehm [1989] täydellisyytenä, tarkkuutena ja johdonmukaisuutena. Vaikka kuvauksia on paljon, yksikään ei ole väärä – ne esittävät hyvin, mitä toimintavarmuudella tavoitellaan.

3.4.2 Kriittinen ylläpidettävyys

Eräiden arvioiden mukaan noin puolet kaikista ohjelmiston elinkaaren kustannuksista kohdistuu ylläpitoresursseihin, joillakin aloilla paljon suurempikin osa, n. 60–80 % informaatiojärjestelmien resursseista [Hanna 1993, Vogel 1996]. Osa kustannuksista voi kuulua toisella tapaa tarkasteltuna normaaliin suunnitteluun ja ohjelmointiin. Vaikka

tarkasteluperusteet vaihtelisivatkin, osuus on huomattava ja haastaa ohjelmistoteollisuuden. Tarvitaan tehokkaita menetelmiä sekä ylläpitotoimintojen riskien vähentämiseksi että laadukkaan ylläpidettävyyden hallitsemiseksi.

Ohjelmiston varhaisissa elinkaarivaiheissa määritellään, mitkä osat ohjelmistoa ovat kriittiset sekä kehitettäessä ohjelmistoa että erityisesti käytön ja ylläpidon aikana. Määrittelemiseen sisältyy myös se, miten merkittävästi muutoksia ja päivityksiä aiotaan tehdä ohjelmiston tai järjestelmän elinaikana. Koodin uudelleen käyttäminen tai suunnitteleminen johtaa vastaavien suunnittelu- ja toteutustarkasteluiden sekä liityntävaatimusten uudelleen käsittelyyn. Ylläpidosta saattaa tulla huonosti toteutettuna vaikea ja työläs, etenkin jos ohjelmiston perusteemaa jatkuvasti sovelletaan. Muutossuunnitteluun kannattaa kiinnittää huomiota riittävän ajoissa.

Swanson [1976] määrittelee ohjelmiston ylläpidon prosessiksi, jossa ohjelmistoa muutetaan sen valmistumisen jälkeen käytön aikana. Hän luettelee kolme ohjelmiston muutosprosessia: korjaavat, mukauttavat ja täydentävät. Korjaavilla prosesseilla ohjelmistoa muutetaan raportoitujen virhetietojen perusteella, mukauttavilla ohjelmistoa sopeutetaan uuteen ympäristöön esimerkiksi, jos laitteisto tai käyttöjärjestelmä vaihtuu, järjestelmä liitetään toiseen systeemiin, tehdään laajennuksia, jne. Täydentävä ohjelmiston ylläpito tarkoittaa uusien vaatimusten toteuttamista tai hienosäätöä esimerkiksi käyttäjän uusien tarpeiden mukaan: rakennetta parannetaan tai suorituskykyä tehostetaan.

Edellä mainittujen kolmen muutosprosessin lisäksi Swanson [1976] mainitsee ennakoitavan muutosprosessin kriittisille ohjelmistoille. Päämääränä kriittisen ohjelmiston ylläpidettävyydessä on jatkuva luotettavuuden arviointi, erityisesti käyttöliittymävirheiden vähentäminen ja ajonaikaisen järjestelmän tarkistaminen ennen kriittisten toimintojen suorittamista. Merkittäviä jo ohjelmistotuotannon aikana huomioon otettavia asioita ja toimenpiteitä ovat seuraavat:

1. Ohjelmakoodin ymmärtäminen. Kriittiset luotettavuutta heikentävät osuudet tunnistetaan.
2. Tarkastellaan, miten välttämättömät muutostyöt voidaan tehdä jo suunnittelun ja toteutuksen aikana. Muutosohjeet dokumentoidaan.
3. Muutostöistä päättäminen. Arvioidaan koodin muutettavuus jo ohjelmistotuotannon eri vaiheissa. Arvioidaan muutosten aiheuttamat vaikutukset.
4. Koodin muuttaminen muutosten jälkeen.
5. Muutosten kelpoistaminen regressiotesteillä.

Yleisiä ongelmia ohjelmiston ylläpidossa aiheuttavat ylläpitoprosessien vaillinainen määrittely sekä puutteet dokumentoinnissa, tekniikoissa, asiantuntemuksessa, kokemuksessa, motivoinnissa ja ajankäytössä. Näitä mahdollisia puutteita tulisi tarkastella kriittisiksi todettujen ohjelmisto-osien käsittelyssä.

Tuotteenhallinta helpottaa ohjelmisto- ja versiomuutosten jäljitettävyyttä ja todennettavuutta. Tuotteenhallinnassa voi sattua virheitä, kun uusi tai päivitetty moduuli yhdistetään jo olemassa oleviin koodinosiin. Yhden virheen korjaaminen saattaa merkitä uuden, testaamattoman polun tai toiminnon syntymistä. Lisäksi muutokset voivat synnyttää uusia virheitä, jotka taas kuormittavat korjaustyötä. Jos ohjelmistolle on tehty luotettavuustarkasteluita, kriittiset muutokset dokumentoidaan analyysiraportteihin ja jäljitetään todennukselle.

Ohjelmiston dokumentaation ylläpitäminen todellisen ohjelmistototeutuksen kanssa on aina ollut työlästä ja puutteellista. Usein tietyt ohjelmistotuotannon vaihedokumentit puuttuvat kokonaan, jolloin muutostyöt koodausvaiheen jälkeen ovat vielä työläämpiä, kuin puuttuvan vaiheen dokumentointi olisi ollut. Suunnitteludokumentaatio voi syntyä vasta koodausvaiheen jälkeen, mutta silloin sen laatiminen voi olla muistinvaraista ja turhauttavaa, onhan koodi jo valmis ja uudet työt odottamassa. Silloin ylläpitovaiheen varsinaiseksi pullonkaulaksi muodostuukin ymmärtää ohjelmakoodia.

Kriittisen ohjelmiston kehitysprosessin ja ylläpidon aikana tulisi kiinnittää erityistä huomiota juuri tuotehallintaan ja suunnitteluohjeisiin. Hyviä suunnitteluominaisuuksia ovat informaation kätkeminen, kriittisten osien eristäminen, yksinkertaisuus ja virhealttiiden ohjelmakielten ominaisuuksien välttäminen. Lisäksi hyviin ohjeisiin kuuluu myös tarkastella, mitä ohjelmiston ei pitäisi tehdä, ei vain sitä, mitä sen pitäisi tehdä.

Järjestelmän ylläpidettävyys mitataan järjestelmän kykynä tehdä korjauksia ja kunnon arviointia. Sitä ei voida mitata niin täsmällisesti kuin toimintavarmuutta ja käytettävyyttä. Kvantitatiivisena mittana käytetään *MTTR*:ää, mutta siihen liittyy eräitä hankalia ominaisuuksia. Esimerkiksi korjaustapa on otettava huomioon. Joitakin järjestelmiä korjaavat käyttäjät, joitakin toisia valmistajat. Järjestelmän itsediagnosointi voi ilmoittaa yksikön virhetilasta, minkä tiedon käyttäjä toimittaa valmistajalle, joka lähettää uuden yksikön tilalle asennusohjeineen. Sisäänrakennettu itsediagnostiikka voi vähentää *MTTR*-arvoa ja siten ylimääräisiä kustannuksia.

3.4.3 Ohjelmiston turvallisuus

Turvallisuudella tarkoitetaan arviota riskin hyväksyttävyydestä. Riskillä tarkoitetaan tehtävän epäonnistumisen todennäköisyyttä ja vakavuutta. Leveson [1986] väittää ohjelmiston olevan turvallinen silloin, kun se järjestelmässä suoritetaan vaaratta. Ohjel-

misto johtaa järjestelmän vaaraan kahdella tavalla: Joko ohjelmiston lähtöarvot tai ajastusvirheet johtavat järjestelmän vaaralliseen tilaan, tai ohjelmisto ei havaitse tai käsittele laitteistovikoja, joihin sen määrittelyn mukaan kuuluisi reagoida.

Leveson [1995] täydentää ohjelmiston turvallisuuden määritelmäänsä toteamalla, että turvallisuuskriittinen ohjelmisto on mikä tahansa ohjelmisto joka suoraan tai epäsuoraan aiheuttaa järjestelmän tilan joutumisen vaaralliseen tilaan.

Vaikutusasteen mukaan ohjelmisto voidaan määritellä ensisijaiseen ja toissijaiseen turvallisuuskriittisyyteen:

- ensisijaisesti turvallisuuskriittiseksi, jos ohjelmiston virhetoiminta voi johtaa järjestelmän, johon ohjelmisto kuuluu, sellaiseen toimintahäiriöön, jonka seurauksena on henkilö- tai ympäristövahinkoja tai tehtävän epäonnistuminen
- toissijaisesti turvallisuuskriittiseksi, jos ohjelmiston häiriö voi epäsuorasti johtaa henkilö- tai ympäristövahinkoihin tai tehtävän epäonnistumiseen.

Ohjelmiston turvallisuutta on joskus pidetty yhtenä osana ohjelmiston toimintavarmuutta, mutta käsitteet eivät ole verrattavissa tällä tavoin. Vaikka turvallisuuskriittinen järjestelmä olisikin luotettava spesifikaatioonsa verraten ja siten sen tulisi toimia virheettömästi, ohjelmistopohjaiset järjestelmät eivät silti välttämättä ole turvallisia. Silloin, kun hyvin luotettava järjestelmä vikaantuu, sen täytyy vikaantua turvallisesti. On tarkasteltava spesifikaatioiden ohi ympäristöön.

Luotettavuuden ja turvallisuuden välinen eroavuus johtaa myös kvantitatiivisissa luotettavuusarvioissa erilaiseen tiedon käyttöön. Epäkäytettävyyttä aiheuttavat vioittumistavat, jotka aiheuttavat esimerkiksi ohjattavan prosessilaitoksen turhia alasajoja, voivat olla turvallisesti eheitä ja eivät siten huononna turvallisuutta.

Toisaalta, ohjelmiston turvallisuus ja ohjelmiston tietoturva ovat käsitteinä lähellä toisiaan. Leveson [1995] korostaa, että kummatkin luotettavuusattribuutit käsittelevät sekä uhkia että vaaroja ja liittyvät kohteen suojelemiseen samalla tavalla mutta suojeltavat menetykset ovat erilaiset. Ohjelmiston turvallisuudella ja tietoturvalla on kuitenkin selkeä ero siinä, että edellinen koostuu tahattomista, hyvää tarkoittavista virheistä, jälkimmäinen aina tavalla tai toisella tahallisista. Tämä ero näiden kahden attribuutin välillä merkitsee huomattavaa eroa niiden virhetyyppien analysoinnissa ja testauksessa.

3.5 Luotettavuusvaatimusten asettaminen

Ohjelmistovaatimukset ovat toimintoja syötteineen, prosesseineen ja tulosteineen sekä käyttöliittymävaatimuksineen, rajoituksineen, arvoalueineen, tarkkuuksineen ja suorituskykyineen. Tässä kuvataan kriittisten vaatimusten kehittämistä ja priorisointia ohjelmistolle sekä esitetään joitakin esimerkkejä vaatimusten täyttymisen kelpoistamistavoista.

3.5.1 Vaatimusasettelun strategia

Luotettavuusvaatimusten määrittäminen on tullut osaksi järjestelmätoimituksiin liittyvää sopimuskäytäntöä, mikä puolestaan edellyttää yhä laajempaa luotettavuustekniikan käsitteiden ja menetelmien tuntemusta sekä tilaajan että järjestelmän toimittajan taholta sopimuksia laadittaessa. Alihankintasuhteessa toimivat laitevalmistajat joutuvat puolestaan yhä enemmän antamaan takeita tuotteittensa ja palvelujensa riittävästä luotettavuuden tuntemuksesta ja tasosta. Luotettavuusvaatimukset määritetään ja allokoidaan asiakas-alihankkija-rajapinnalle. Toinen tätä yleistävämpi tapa on allokoida vaatimukset toteuttamisen eri vaiheille tuotosdokumentaatiota myötäillen.

Luotettavuusvaatimusten määrittäminen voi tapahtua kolmella tavalla:

1. Ohjattava kohde asettaa vaatimukset, joita allokoidaan kohti yksityiskohtaista komponenttitasoa. Komponenteille määräytyvät allokoinnin ja suunnittelun tuloksena vaatimukset, jotka komponenttien on täytettävä.
2. Järjestelmän luotettavuusominaisuudet määräytyvät suoraan osatoimintojensa luotettavuusominaisuuksien perusteella. Toiminnallisessa hierarkiassa alimmalla tasolla ovat yksittäisten laitevalmistajien laitteet ja komponentit sekä kunnossapitoyritysten palvelut, jotka viime kädessä määräävät, mille tasolle järjestelmän luotettavuusominaisuudet asettuvat. Komponenteilla on tietyt luotettavuusarvot, joita ei voida muuttaa. Esimerkiksi COTS (Commercial Off The Self) -ohjelmisto- tai laitteistokomponentit ovat valmiita tuotteita, joilla on tai ainakin pitäisi olla tietyt valmistajan ilmoittamat luotettavuusarvot. Komponentit kootaan järjestelmäksi. Erityisesti laitteiston kvantitatiivisessa luotettavuusarvioinnissa usein vain lasketaan järjestelmän jokin luotettavuusarvo (toimintavarmuus, käyttövarmuus tai turvallisuuden eheys) ja lopuksi päätellään sen riittävyydestä ilman erityisiä suunnitteluun vaikuttavia vaatimusarvoja.
3. Integroidaan kaksi edellistä määrittämistapaa iteroivalla tekniikalla. Määritetään luotettavuusvaatimukset, jotka allokoidaan määrittely-, suunnittelu- ja toteutusvaiheissa kohti komponenttitasoa. Suunnittelu- tai toteutustasolla voidaan havaita, että sopivaa suunnitteluratkaisua tai val-

mista komponenttia ei ole olemassa, joten tietty kehitysvaihe uudistetaan, kunnes päästään tasapainoiseen ratkaisuun. Usein luotettavuusvaatimukset ovat järjestelmätasoilla vakioita, niin että niiden muuttaminen ei ole mahdollista.

Luotettavuus voidaan nähdä myös yksittäisen ohjelmistovalmistajan oman toiminnan kehittämisen välineenä. Tällöin luotettavuusvaatimukset tai allokoidut vaatimukset ovat yrittäjän oman strategisen liikesuunnittelun määrittämiä välitavoitteita, joihin pyrkimällä pyritään varmentamaan esimerkiksi suunnannäyttäjän imago kyseisillä markkinoilla.

Järjestelmän toimintoihin, esimerkiksi laitteeseen tai ohjelmistokomponenttiin, kohdistettavien luotettavuusvaatimusten voidaan katsoa olevan lähtöisin kahdesta eri perustilanteesta: 1) tilaajan toimittajalle asettamat vaatimukset tai 2) laitevalmistajan itsellensä asettamat vaatimukset. Edellinen liittyy tilaajan ja toimittajan väliseen sopimukseen järjestelmän, esimerkiksi ohjelmoitavan logiikan, luotettavuusvaatimusten tasosta ja usein myös vaatimusten todentamisen keinoista. Tämä on seurausta tilanteesta, jossa investointihyödykkeitä valmistava teollisuus yhä enemmän ulkoistaa monia yrityksen toimintoja, kuten erikoiskomponenttien valmistusta tai ohjelmistotuotantoa, jotta yritys voisi keskittyä ydinliiketoimintaansa. Haittana voidaan pitää keskitetyn kontrollin menetystä tietyiltä tuotantolaitoksen luotettavuusominaisuuksien hallinnan osilta. Jälkimmäinen perustilanne vastaa yksittäisen yrityksen strategiaan liittyvää tuotteen luotettavuusominaisuuksien kehittämistavoitetta. Luotettavuusvaatimusten määrittelijä on yrityksen johto, jonka yritysvisiossa luotettavuus näyttelee merkittävää osaa.

Luotettavuusvaatimusten määrittelyssä turvallisuuden käsittely poikkeaa selvästi muista luotettavuustekijöistä. Tähän vaikuttavat olennaisesti viranomaismääräyksiin liittyvät vaatimukset, esimerkkinä Seveso-direktiivi, joka asettaa omat turvallisuuden analyysivaatimuksensa, jotka koskevat prosessiteollisuutta. Poikkeavuuteen vaikuttavat kuitenkin ensisijaisesti turvallisuuden määrällistämiseen liittyvät ongelmat. Riskianalyyseissä turvallisuuden mitta on riski, joka vaivoin muuntuu taloudellisiin mitta-asteikkoihin. Päätösanalyysin puitteissa voidaan sekä riski että markka muuntaa yhteiseksi yksiköksi johdonmukaisella tavalla. Muunnos on kuitenkin subjektiivinen ja sisältää yksityiskohtaista arvojen mallintamista, mikä edellyttäisi kokonaan uudenlaisen lähestymistavan omaksumista investointilaskelmia tehtäessä.

3.5.2 Luotettavuusprofiili

Ohjelma on kriittinen, jos sen virhetoiminta voi aiheuttaa riskin henkilöille, ympäristölle, omaisuudelle tai toiminnan keskeytykselle. Ohjelmiston kriittisyystarkastelun tavoitteena on merkitä tietyt vaatimukset kriittisiksi myöhempiä jatkotoimia varten. Tar-

kastelulla selvitetään myös määrittelyn täsmällisyyttä, jolla kriittisissä tapauksissa tarkoitetaan sitä, että suunnittelu voi kattavasti ja yksiselitteisesti pohjautua siihen.

Kriittiset vaatimukset voivat olla joko ylhäältä alas tai alhaalta ylös johdettuja. Edelliset perustuvat järjestelmävaatimuksiin, joihin ne on saatettu kehittää vaara-analyysien tuloksista, jälkimmäiset perustuvat suunnittelun ja toteutuksen aikaisiin tarkastelutuloksiin. Käytännössä kyse on yleensä aina näiden kahden alkulähteen tietynlaisesta yhdistämisestä, jossa iteratiivisesti aluksi määritellyt vaatimukset modifioidaan suunnitteluratkaisujen tarkasteluista. Vaatimukset voidaan myös määrittää ja priorisoida tietyssä ohjelmiston elinkaarivaiheessa, esimerkiksi määrittelyvaiheessa, jossa vaatimus vaatimukselta tarkastellaan ohjelman mahdollisia kielteisiä seurausvaikutuksia ja ennustetaan niihin johtavien virhetoimintojen esiintymistodennäköisyyttä. Tällä menettelyllä jokainen vaatimus ja toiminto priorisoidaan tiettyyn luotettavuusprofiiliin.

Ohjelmiston kriittisyyttä tarkastellaan tässä neljässä vaiheessa:

1. Määritetään yritys- tai tuotekohtaiset vakavuustasot.
2. Asetetaan järjestelmää kuvaavat luotettavuusvaatimukset.
3. Asetetaan ja allokoidaan vaatimusten luotettavuusprofiili.
4. Määrätään sopiva kelpoistusmenettely tiettyjen vaatimusten täyttymisen osoittamiseksi.

Kriittisyystasot voidaan ilmaista joko luotettavuusattribuuteittain tai yhtenäisenä kriittisyyttä ilmaisevana luokituksena. Luokat voivat sisältää joko esimerkiksi arvot korkea, kohtalainen ja matala tai numeeriset arvot 1–N. Numeerista arvoa käytetään usein elektroniikalle sovelletussa riskinmääritysmenetelmässä RPN[1], jossa taso ilmaistaan kolmen indeksin, vakavuus-, havaittavuus- ja esiintyvyystekijöiden, tulona. Menetelmässä vakavuus ja esiintyvyys ovat yleisesti käytetyt riskitekijät, harvemmin käytetty havaittavuustekijä ilmaisee, miten hyvin vaara tai virhe voidaan paljastaa kehitysprosessin aikana mm. analyyseilla ja testeillä sekä tuotteessa suunnitteluratkaisuin, esim. itsediagnostiikka- ja pariteettibittitekniikoilla. Havaittavuustekijä kuuluu yleensä osana esiintyvyystekijään. RPN-menetelmästä löytyy runsaasti viitteitä internetissä, joista monipuolisin lienee http://www.fmeca.com vielä keväällä 2000. Siinä kullekin kolmelle riskitekijälle annetaan asteikko 1–10, jolloin suurimmaksi riskiluvuksi saadaan 1 000.

Tämän julkaisun tavoitteiden kannalta esitetään pelkistetty riskitason määrittelymenetelmä, jota parhaiten kuvaavat taulukot 1–3. Riskitaso (Taulukko 3) muodostuu vaiku-

[1] Risk Priority Number

tusluokista (Taulukko 1) ja tarkasteltavan virhetoiminnon esiintymistiheysasteesta (Taulukko 2). Riskitasot voidaan määrittää halutun riskinoton mukaisesti, pääasia on, että menettelyä sovelletaan johdonmukaisesti. Pelkistetympää määrittelymenetelmää kannattaa muutenkin käyttää ensiksi ja siitä saatujen kokemusten perusteella hienontaa asteikkoa useampaan tasoon, kuten RPN-menettelyssä.

Taulukko 1. Vaaratapahtuman vaikutusluokitus.

Kuvaus	Luokka	Vaikutusluokan määrittely
Katastrofaalinen	A	Kuolema, järjestelmän menetys, tai vakava ympäristövahinko
Kriittinen	B	Vakava loukkaantuminen, vakava sairastuminen, suurehko järjestelmä- tai ympäristövahinko
Vähäinen	C	Lievä loukkaantuminen, lievä sairastuminen, pieni järjestelmä- tai ympäristövahinko
Merkityksetön	–	Vähempi kuin lievä loukkaantuminen tai sairastuminen, tai vähempi pieni järjestelmä- tai ympäristövahinko

Taulukko 2. Tapahtuman esiintymistiheyden asteikko.

Kuvaus	Luokka	Esiintymistiheysasteen määrittely
Erittäin korkea	I	Tapahtuu hyvin usein
Korkea	II	Tapahtuu muutaman kerran vuodessa
Kohtalainen	III	Tapahtuu muutaman kerran kohteen elinaikana
Alhainen	IV	Epätodennäköistä kohteen eliniässä
Erittäin alhainen	V	Käytännössä ei tapahdu eliniän aikana

Korkeilla riskitasoilla tarvitaan perusteellisia analyyseja ja testauksia sekä myös painottamista suunnitteluratkaisujen valinnassa. Jos tietyn ohjelmistotoiminnon riskitaso on todettu liian korkeaksi, suunnittelu- tai arviointitoimin alennetaan riskiä siedettävälle tasolle (’matala’ Taulukossa 3). Riskin vähentäminen tapahtuu yleensä esiintymistiheyden arvioita pienentämällä, harvemmin kyetään seurausten vakaavuutta alentamaan. Esiintymistiheyden pienentämisen arvioiminen ei myöskään ole kovin helppoa, useimmiten tarvitaan kokemusperäistä tietoa aikaisemmista projekteista käytettyjen suunnit-

teluratkaisujen ja laadunvarmistustoimenpiteiden onnistumisesta. Täsmällisin menetelmä on suunnittelutuloksista tehtävä matemaattinen luotettavuusarviointi, joka perustuu asiantuntija-arvioihin ohjelmistovirheiden esiintymisestä. Tällä menetelmällä kyetään arvottamaan erilaiset riskiä vähentävät toimenpiteet siten, että voidaan määrittää, mitä menetelmiä kannattaa käyttää, kun alustava riskitaso on ’hyvin korkea’, ’korkea’ tai ’matala’.

Taulukko 3. Riskitasot muodostuvat mahdollisen virhetoiminnon vaikutusluokasta ja esiintymistiheydestä.

Esiintyminen	**Vaikutusluokka**			
	A	**B**	**C**	**–**
I	Hyvin korkea	Hyvin korkea	Korkea	Kohtalainen
II	Hyvin korkea	Korkea	Kohtalainen	Matala
III	Korkea	Korkea	Kohtalainen	Matala
IV	Korkea	Kohtalainen	Matala	Matala
V	Kohtalainen	Matala	Matala	Matala

Esiintymistiheyden arvioinnin hankaluus voi johtua myös tuotteen laatutekijöistä. Esimerkiksi määrittelyvaiheessa esiintymistiheyden arviointi voi olla vaikeaa, jos määrittely ei ole kovin yksiselitteinen eli tietyt vaatimukset voidaan tulkita useammalla tavalla. Tällaisessa tapauksessa määrittelyn tärkein laatukriteeri ’täsmällisyys’ ei täyty. Joko vaatimuksesta on informoitava lisää, tarkennettava tai karsittava sitä.

Usein ohjelmistotuotannon prosesseissa vaatimuksia ei lainkaan dokumentoida. Tämä tulisi kuitenkin tehdä aina silloin, kun ohjelmistotuote vaikuttaa tai tiedetään kriittiseltä jonkin luotettavuusattribuutin osalta. Vaatimukset merkitään tunnistenumeroilla, mikä helopottaa jäljittämistä.

Kriittisyyden mukaista jaottelua ei yleensä oteta ohjelmistotuotannon eri vaiheissa riittävästi huomioon, mistä seuraa, että ohjelmistovirheet tai laitteistoviat voivat aiheuttaa häiriöitä eri puolilla järjestelmää. Yksinkertainen ohjelmiston riskiluokitus voisikin perustua eristämiseen, vikasietoisuuteen, yksinkertaisuuteen sekä analyysien ja testien riittävyyteen:

1. Ei-kriittiset toiminnot. Jos ohjelmisto on selvästi erotettu kriittisistä toiminnoista ja muista ohjelmistoista, ei-kriittinen ohjelmisto ei todennäköisesti aiheuta vakavia vaikutuksia.
2. Kriittiset toiminnot. Ohjelmiston erottamisen ja energian rajoittamisen takia vaikutukset ovat rajoitettuja.
3. Hyvin kriittiset toiminnot. Tuhoisat vaikutukset ovat mahdollisia, mutta epätodennäköisiä, jos sekä laitteisto että ohjelmisto ovat vikasietoisia ja riittävän yksinkertaisia, että ne voidaan analysoinnein ja testein kelpoistaa.

Ohjelmistolle kohdistetut kriittiset vaatimukset muunnetaan usein toiminnallisiksi vaatimuksiksi kehitysprojektin kuluessa.

Ohjelmiston kriittisille luotettavuusvaatimuksille pätevät esimerkiksi Taulukon 4 esittämät säännöt:

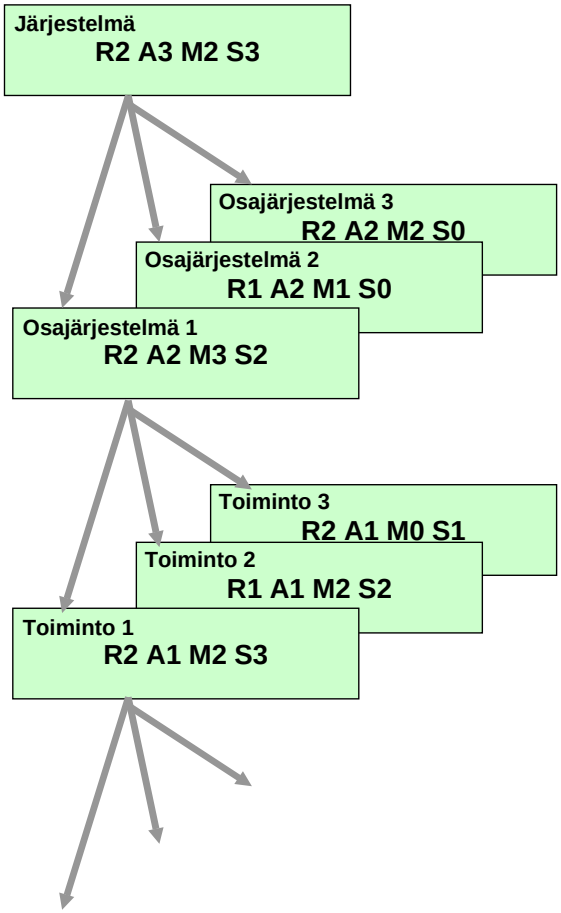
Taulukko 4. Esimerkki ohjelmiston luotettavuusvaatimuksista.

Tunniste	Geneerinen luotettavuusvaatimus
1	Kriittisten virhetoimintojen ajonaikainen automaattinen tunnistaminen, eristäminen ja palautuminen normaalitoimintaan on järjestettävä.
2	Ohjelmistovirheen eteneminen kriittisiin moduuleihin on estettävä.
3	Järjestelmän on toivuttava virhetoiminnosta.
4	Toiminnon suunniteltu elinaikavaatimus on N vuotta.
5	Toiminnon keskeytymätön käyttöaika on N kuukautta.
6	Kriittiset virhetilanteet ja -toiminnot on havaittava.
7	Diagnosointiprosesseja on valvottava.
8	Käytön aikaisista virhetoiminnoista on huolehdittava.
9	Tietyt laitteistoviat on hallittava ohjelmistolla.
10	Yksittäisvika saa aiheuttaa tietyn tason vioittumista <0.001 todennäköisyydellä.
11	Operaattori käynnistää varmistukset kaikissa mahdollisissa asianmukaisissa vikatilanteissa.
12	Virhetoiminnon jälkeiset toimenpiteet tulee määrittää.
13	Ohjelman suoritusta on valvottava.
14	Muistin kuntoa on valvottava.

Luotettavuusvaatimusten asettaminen on iteroiva prosessi, mikä voi olla osana suunnitteluprosessia (ks. Taulukko 5). Vaatimusmäärittelyn ja ohjelmiston suunnittelun välinen raja on muutenkin epäselvä.

Luotettavuustavoitteet saadaan turvallisuusattribuutille ohjattavalle kohteelle tehdyistä vaara-analyyseista. Tavoitteet spesifioituvat kaikille järjestelmän osille ja toiminnoille. Vastaavasti tulisi määrittää toimintavarmuus-, käytettävyys- ja ylläpidettävyysattribuuttien tavoitteet ja niistä johdetut yksilöidyt vaatimukset siten, että koko luotettavuus saadaan kuvattua tiettynä luotettavuusprofiilina. Luotettavuusprofiili voidaan esittää esimerkiksi luokitteluna tietyn systeemin mukaan: RrAaMmSs, missä pieni kirjain tarkoittaa vastaavan attribuutin luokkaa tai tasoa (esimerkiksi R2A3M2S3, Kuva 5).

.



Kuva 5. Esimerkki luotettavuusprofiilin allokoimisesta. Järjestelmän luotettavuusprofiili allokoidaan osajärjestelmille ja toiminnoille.

Luotettavuusprofiilin hyödyntäminen perustuu kompleksisilla järjestelmillä hyödynnettyyn ylhäältä-alas-tekniikkaan (olio-ohjelmointi yms.), jossa järjestelmä jaetaan ensin osajärjestelmiin ja edelleen komponentteihin. Ylhäältä-alas-tekniikkaa hyödynnetään [Deswarte et al. 1999]

- kompleksisuuden hallitsemisessa
- merkittävien ominaisuuksien keskittämisessä vain muutamaan kohteeseen mieluimmin kuin hajottamisessa kaikkialle
- eri projektiryhmissä kehittäessä erityyppisiä komponentteja (mm. ohjelmisto ja laitteisto)
- alihankintojen käytön parantuessa niihin kohdistuvien vaatimusten täsmentyessä
- toteutettaessa komponentteja valmiista osista, esimerkkinä Commercial Off The Shelf (COTS) -tuotteet.

Luotettavuusprofiilin allokointi vaihe vaiheelta noudattaa luonnollisella tavalla ylhäältä-alas-tekniikkaa, koska toiminnalliset ja ei-toiminnalliset vaatimukset luotettavuustoimintoja ja -vaatimuksia myöten jaetaan samalla tekniikalla samoille kohteille. Kuvan 5 esityksessä on huomattavaa, että turvallisuusattribuutti on keskitetty vai 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.

Luotettavuusprofiilia allokoitaessa ja pienennettäessä hyödynnetään jokaista vaatimusten riskitasoa määrittämällä sopiva todennus- ja kelpoistusmenettely, jolla osoitetaan tiettyjen vaatimusten ja ominaisuuksien saavuttaminen. Taulukossa 5 on joukko esimerkkejä luotettavuusvaatimusten kelpoistustekniikoiden käyttämisestä riskinvähennykseen, ks. kohdasta 5.1 riskinvähennyksen tekniikoista yleensä. Seuraava esimerkki liittyy Taulukon 4 esimerkkiin ohjelmiston luotettavuusvaatimuksista.

Taulukko 5. Suunnitteluprosessit luotettavuuden parantamiseksi esimerkin (ks. Taulukko 4) esittämille vaatimuksille.

Tunniste	Luotettavuuden suunnittelu
1	Tunnistetaan kaikki ne toiminnot, joiden virhetoiminnot voivat aiheuttaa kriittisiä vaikutuksia. Vaikutusten suuruudet ja esiintymistodennäköisyydet arvioidaan (Taulukot 1 ja 2). Toiminto merkitään tiettyyn riskitasoon (Taulukko 3) tai luokitteluna voi olla esimerkiksi seuraava: 1) turvallisuuskriittiset virhetoiminnot, 2) totaaliseen tai 3) osittaiseen tehtävämenetykseen johtavat virhetoiminnot. Tunnistaminen suoritetaan vika-, vaikutus- ja kriittisyysanalyysilla (ohjelmiston FMECA, ks. Luku 5).
2	Tunnistetaan laitteiston ja ohjelmiston vuorovaikutteiset virhetilanteet ja -toiminnot laitteiston ja ohjelmiston vuorovaikutusanalyysilla (HSIA, ks. Luku 5). Tunnistetaan niihin johtavat syyt ja poistetaan virheet.
3	Järjestetään virhetoiminnoista toipuminen joko 1) redundantilla toiminnolla, 2) uudelleenkonfiguroinnilla (mm. reset-toiminta) tai 3) vikaturvallisella toiminnolla.
4	Todennetaan toiminnon elinaikavaatimus vikapuuanalyysilla.
5	Todennetaan toiminnon elinaikavaatimus ja keskeytymätön käyttöaikavaatimus vikapuuanalyysilla.
6	Tunnistetaan luotettavuusanalyyseilla (ohjelmiston FMECA tai FTA) kriittiset virhetilanteet ja -toiminnot, jotka järjestelmän automaattisesti diagnosoi.
7	Operaattori todentaa diagnostiikkatoiminnot käytönaikaisin testauksin.
8	Kirjataan virheiden siedettävyysvaatimukset ylläpito-ohjeisiin.
9	Tunnistetaan luotettavuusanalyyseilla kriittiset laitteistoviat ja suositellaan ohjelmistotoimintoja riskin vähentämiseksi.
10	Arvioidaan yksittäisvian esiintymistiheys vikapuuanalyysilla.
11	Tunnistetaan ne virhetoiminnot, joiden käsittelyyn tarvitaan operaattoria tai valvojaa.
12	Automaattisesti tai operaattorin käskystä toteutetaan vikaturvallisuustoiminnot virhetoiminnon havaitsemisen jälkeen.
13	Tunnistettava ajastustekijät, jotka vaikuttavat käyttöjärjestelmän vahtikoiran virkistämiseen.
14	Muistin hallintajärjestelmä tarkistaa jatkuvasti muistin kuntoa (virhesietoisuus ja testit).

4. Luotettavuuden asettamat vaatimukset ohjelmistotekniikalle

Ohjelmistotuotanto voidaan jakaa esimerkiksi projektinhallintaan, laadunvarmistukseen ja ohjaukseen, tuotteenhallintaan, vaatimustenhallintaan, todennuksiin, dokumentointiin, teknisiin katselmuksiin, vaatimusten jäljittämiseen, ulkopuolisiin tarkistuksiin jne. Ohjelmiston kehitysprosessiin kuuluu useita vaiheita tietyn elinkaarimallin mukaan. Tunnetuin elinkaarimalli lienee vesiputousmalli, jota usein käytetään referenssimallina tiettyjen vaatimusten liittämiseksi kehitysprosessiin. Vesiputousmalliin kuuluvat ainakin määrittely, suunnittelu ja ohjelmointivaiheet. Lisäksi usein liitetään mukaan esitutkimus, arkkitehtuurisuunnittelu, integrointi- ja testausvaiheet sekä niitä seuraavat käyttöönotto- ja ylläpitovaiheet.

Luottamusta ohjelmiston riittävään luotettavuuteen voidaan kasvattaa arvioimalla liittyvää dokumentaatiota tai arvioimalla prosesseja, joilla ohjelmisto kehitetään. Luotettavuutta parannetaan erityismenetelmin, kuten kohdassa 3.3 todettiin. Laatusuunnittelulla tavoitellaan mahdollisimman virheetöntä ohjelmistoa. Tietyn luokituksen, resurssien ja asiakkaan spesifististen vaatimusten puitteissa tehdään mahdollisimman luotettava ja turvallinen ohjelmisto.

Tärkein – tosin ei riittävä – ohjelmiston luotettavuutta parantava keino on kuitenkin huolellisuus kaikissa mahdollisissa vaiheissa ja tehtävissä koko ohjelmiston eliniän aikana. Tässä luvussa tarkastellaan luotettavaa ohjelmistokehitystä muutamassa keskeisessä vaiheessa. Samalla tiedostetaan, mitä voidaan vaatia luotettavuusvaatimusten kelpoistamismenetelmältä.

4.1 Ohjelmiston vaatimusmäärittely

Ohjelmiston määrittelyvaihetta edeltävää järjestelmän vaatimusmäärittelyä kutsutaan etenkin sulautettujen ohjelmistojen yhteydessä usein järjestelmäsuunnitteluksi. Siinä kuvataan kokonaisuutta: kokonaisarkkitehtuuri, eri osien tehtävät, laitteiston ja ohjelmiston työnjako. Usein ohjelmiston vaatimusmäärittely- ja järjestelmän vaatimusmäärittelyvaihetta ei selkeästi eroteta toisistaan.

Määrittelyvaiheessa määritetään loppukäyttäjän (asiakkaan, järjestelmätason) tarpeet vaatimuksina sekä kuvataan laite- ja ohjelmistoympäristöt. Vaatimukset dokumentoidaan järjestelmän vaatimusdokumenttiin, jonka pohjalta ohjelmistosuunnittelija määrittää toteutettavan ohjelmistotuotteen. Tuloksena syntyvää dokumenttia kutsutaan toiminnalliseksi määrittelyksi. Se sisältää kuvaukset ohjelmistojen toiminnoista, eitoiminnalliset vaatimukset sekä reunaehdot ja rajoitteet. Toimintoihin kuuluvat ohjel-

mistoilla toteutettavat ominaisuudet, käyttöliittymät ja kommunikoinnit muiden järjestelmien kanssa. Ei-toiminnallisia vaatimuksia ovat mm. vasteaika, teho ja luotettavuus. Rajoituksia ovat mm. käytetty ohjelmointikieli ja muistikapasiteetti.

Korkean luotettavuuden tavoittelu on kallista. Siksi onkin syytä kohdentaa sekä luotettavuustarkastelut että ohjelmistotuotannon kehitys-, todennus- ja kelpoistusprosessit tiettyihin kriittisiksi todettuihin kohteisiin. Priorisoidaan vaatimukset.

Priorisointitarkastelussa tunnistetaan ja järjestetään tiettyihin riskiluokkiin kaikki ne ohjelmiston vaatimusdokumentissa määritellyt vaatimukset, jotka voivat aiheuttaa merkittävää luotettavuuden alentumista ohjelmiston virhetoimintojen takia. Virhetoiminnon kriittiset vaikutukset ovat mahdollisia, jos vaatimuksia ei ole oikein määritetty tai toteutettu. Priorisoinnin tavoitteena on siten myös ohjata ja painottaa suunnittelua, toteutusta ja ylläpitoa luotettavuutta parantavin keinoin, analysoinnein ja testein.

Priorisointitarkastelun kuvaus viittaakin määriteltyjen vaatimusten hyvään *hallittavuuteen*, *määritettävyyteen* ja *suoritettavuuteen*. Ilman näitä ominaisuuksia priorisoinnilla ei saavuteta sille asetettuja tavoitteita.

Hyvältä vaatimusten *hallittavuudelta* edellytetään ensi sijassa sitä, että ohjelmistovaatimukset vastaavat täydellisesti järjestelmätason vaatimuksia ja alkuperäisiä ohjelmistoon kohdistuvia mahdollisesti kirjoittamattomiakin tarpeita. Lisäksi vaatimusten toteutus, ohjelmistokomponentit, vastaavat täydellisesti ohjelmistovaatimuksia. Vaatimustenhallinnan tärkein tehtävä on varmistaa, että ohjelmistokomponentit jäljittyvät alkuperäisiin vaatimuksiin. Lopputuotteessa on oltava kaikki halutut ominaisuudet ja vain ne, sillä ylimääräiset ominaisuudet kasvattavat riskiä jonkun luotettavuusattribuutin kohdalla.

Kaikkia alkuperäisiä vaatimuksia ei heti alussa tunneta vaan ne täydentyvät ja täsmentyvät projektin edetessä. Vaatimusten täydentäminen projektin kuluessa ei varsinaisesti ole ongelmallista luotettavuudessa vaan vaatimusten muuttaminen ja täsmentäminen, sillä jäljitettävyys kuormittuu heikentäen hallittavuutta. Eräs mahdollisuus parantaa tilannetta on tunnistaa muutosherkät tai vaikeasti ylläpidettävät vaatimukset.

Ohjelmistotuotannon jokaisen elinkaarivaiheen päätteeksi todennetaan vaatimusten toteutuminen vertaamalla tuotosdokumentteja vaiheen syötedokumentteihin. Todennustuloksista voidaan osoittaa kohdat, jotka toteuttavat tietyn alkuperäisvaatimuksen, ts. vaiheen tulokset ovat jäljitettävissä taaksepäin vaatimuksiin. Eteenpäin jäljitettävyydessä varmistetaan, että kaikki alkuperäiset vaatimukset on toteutettu suunnittelussa ja koodauksessa. Tarkastelulla varmistetaan, että vaatimuksissa on kaikki välttämättömät toiminnot ja rajoitteet sekä että ne eivät sisällä tarpeettomia vaatimuksia.

Toiminnallisten vaatimusten jäljittäminen on paljon helpompaa kuin ei-toiminnallisten vaatimusten. Usein ei ole mielekästä osoittaa kaikkien ei-toiminnallisen vaatimusten, reunaehtojen tai rajoitteiden jäljitettävyyttä alkuperäisiin vaatimuksiin, riittää kun priorisointitasoltaan merkittävimmät arvioidaan.

Hyvin määritetyt vaatimukset on helppo arvioida määrittelyvaiheessa ja niiden toteuttaminen on helppoa suunnittelu- ja koodausvaiheissa. *Määritettävyydellä* tarkoitetaan sitä, miten helposti asiasisältö saadaan ilmaistua formaaleiksi tai ei-formaaleiksi toteuttamis- ja arviointikelpoisiksi vaatimuksiksi. Termiä lähellä on myös vaatimusten ylläpidettävyys, jolla tarkastellaan vaatimusten muunnettavuutta uuteen tarkoitukseen siten, että valmis loppu- tai vaihetuote ei vaadi suuria toteuttamis- tai arvioimisponnisteluita. Ylläpidettävyyteen kuuluu myös uusien vaatimusten ristiriidaton lisääminen.

Vaatimusmäärittelyn todentamisessa tavoitteena on arvioida ohjelmistovaatimusten täydellisyyttä, oikeellisuutta, ristiriidattomuutta ja todennettavuutta (tai testattavuutta). Jokainen vaatimus arvioidaan sekä itsenäisenä osana että yhdessä kokonaisuuden kanssa. Täydellisyydellä tarkoitetaan ensisijaisesti sitä, että kyetään kahden peräkkäisen vaiheen välillä arvioimaan, että kaikki vaaditut elementit ovat olemassa ja oikein toteutetut. Lisäksi vaatimusmäärittelyvaiheessa vaaditaan, että seuraavan vaiheen toteuttaja, suunnittelija, ymmärtää vaatimuksen täydellisesti. Hyvin määritellyt vaatimukset ovat hyvä lähtökohta suunnittelulle.

Vaatimusmäärittelyn analyysitekniikoita ovat ohjausvuoanalyysi, informaatiovuoanalyysi ja toimintoanalyysit. Vaatimusten ohjausvuoanalyysi tarkastelee ohjelmistotoimintojen suoritusjärjestystä tavoitteena tunnistaa puuttuvat ja ristiriitaiset vaatimukset. Informaatiovuoanalyysi tarkastelee toimintojen ja tiedon välisiä suhteita väärien, virheellisten, puuttuvien ja ristiriitaisten rajapintamäärittelyjen tunnistamiseksi. Toimintoanalyysi mallintaa ja arvioi ohjelmistokomponentin ominaisuudet suorituskyvyn, järjestelmäominaisuuksien ymmärrettävyyden ja toteutettavuuden arvioinnin suhteen.

Hyvältä *suoritettavuudelta* vaaditaan ensisijaisesti ajoitus- ja mitoitusvaatimusten riittävyyttä ohjelmistolle asetettuihin rajoitteisiin verraten. Rajoitteita ovat maksimisuoritusaika, kriittisten funktioiden suoritusaika ja muistin enimmäiskäyttöaika. Tavoitteena on arvioida suoritusaikaan ja muistin allokointiin liittyvien ohjelmistovaatimusten täsmällisyyttä ja toteutusmahdollisuutta erityisesti ääriolosuhteissa.

4.2 Ohjelmistosuunnittelu

Suunnitteluvaiheessa suunnitellaan yleensä kaksivaiheisesti määrittelyn kuvaamien toimintojen toteutus. Suunnitteluvaihe käsittää arkkitehtuurisuunnittelun ja detaljisuunnit-

telun. Suunnitteluvaiheita voi olla muitakin. Vaiheiden tuloksena syntyy kaksi perusspesifikaatiota, toinen komponenteille ja toinen moduuleille. Alustava testispesifikaatio laaditaan myös suunnitteluvaiheissa.

Virheet etenevät virhetiloiksi erityisesti silloin, kun ohjelmistoa integroidaan. Joihinkin tapahtumiin järjestelmä antaa vastineeksi tilanvaihdon. Nämä tapahtumat kehittyvät joko sisäisesti ohjelmaosuudessa tai tulevat ympäristöstä. Tilanvaihdot vaikuttavat järjestelmän käyttäytymiseen joko aiheuttamalla sisäisiä tapahtumia tai ulkoisia vastineita ympäristöön. Ympäristöä voidaan pitää myös suljettuna järjestelmänä, kuten ohjelmistoakin. Syötetiedot (tapahtumat) voivat aiheuttaa ylikuormitustilan, joka johtaa järjestelmän tietynlaiseen vikatapahtumaan; virheet (tapahtumat) johtavat virhetiloihin, jotka johtavat virhetoimintoihin; vaaratapahtumat johtavat turvallisuuden virhetilanteisiin, jotka voivat johtaa onnettomuustapahtumiin. Se mikä on virhe, virhetilanne tai virhetoiminta riippuu siitä, mitä kohtaa järjestelmähierarkiaa tarkastellaan.

Ohjelmiston suunnittelu ja kehitys vaihtelevat tapauksesta riippuen. Vaihteluun vaikuttavat sovellus, valittu kehitysprosessi ja tavoiteltu arkkitehtuuri. Esimerkiksi ohjelmoitavassa logiikkajärjestelmässä sovellusohjelmointi (tikapuulogiikka tai toimintolohkot) on olennaisilta osiltaan konfigurointia. Vaihtelevuudesta huolimatta tulisi kaikessa ohjelmoinnissa edetä rakenteellisesti seuraavalla tavalla:

- järjestää koko ohjelmisto selkeästi tunnistettaviin modulaarisiin rakenteisiin
- tunnistaa järjestelmään läheisesti liittyvät eri moduulit
- suojata ohjelmisto erilaisin tarkistuksin virheellisiltä tietojen syötöiltä, joita tulee muualta järjestelmästä tai käyttäjiltä
- hyödyntää mahdollisimman pitkälle todennettua ja hyvin toimivaksi todettua ohjelmistoa, komponenttia tai moduulia
- pyrkiä helposti modifioitaviin rakenteisiin.

Yleisiä suunnitteluvaatimuksia ovat mm. seuraavat:

- Suunnittelukuvaukset esitetään hyvin määritellysti ja ristiriidattomasti.
- Ylläpidettävyyteen ja todennettavuuteen kiinnitetään huomiota koko suunnittelun aikana.
- Ohjaus- ja tietovuokaaviot liitetään suunnitteluaineistoon.
- Olemassa oleva ohjelmiston osuus kuvataan ja soveltuvuus tarkastetaan.

Arkkitehtuurissa tai arkkitehtuurisuunnittelussa järjestelmä jaetaan osiin eli moduuleihin. Tietokoneen arkkitehtuuri käsittää kolme osaa: laitteistoarkkitehtuurin, ohjelmisto-

arkkitehtuurin ja niiden väliset kartoitukset. Lisäksi ohjelmistokehitys voi kohdentua perus- tai sovellusohjelmiston toimintoihin. Edellisiä ovat mm. käyttöjärjestelmä ja toimilohkot, jälkimmäisiä perusohjelmistoja hyödyntävät sovellusohjelmistot.

Perus- ja sovellusohjelmistoja tukevia ohjelmistoja ovat mm. tietokantajärjestelmät, tietoliikenneohjelmistot ja kääntäjät. Sovellusohjelmistoja ovat teknistieteelliset, tietämyspohjaiset, kaupallis-hallinnolliset, testi- ja analyysiohjelmistot, prosessiautomaatiojärjestelmien ohjelmistot ja yleensä sulautettujen järjestelmien ohjelmistot jne.

Sovellusohjelmiston toiminnot rakentuvat moduuleista, jotka voivat kuulua useaan eri toimintoon. Siten yhden moduulin virhe voi edetä usean muunkin ohjelmiston virhetoiminnoksi. Lisäksi sovellusohjelmiston toiminnot voivat käyttää useita perusohjelmiston toimintoja, joiden virhe voi aiheuttaa usean sovellusohjelmiston virhetilan. Järjestelmätasolla tarkasteltuna kukin sovellusohjelmiston toiminto vastaa tiettyä järjestelmän toimintoa. Siten sovellusohjelmiston virhe voi aiheuttaa järjestelmän tämän tietyn toiminnon vikaantumisen.

Sovellusohjelmiston moduulit tai peruskomponentit voidaan ryhmitellä esimerkiksi seuraaviin kokonaisuuksiin: tietyt sovellustoiminnot, aritmetiikka, tietomuunnokset, logiikka, mittaukset, ohjaukset, asetusarvot, yleismoduulit jne.

Tarkoitus on todentaa, että ohjelmistosuunnittelut oikein ja ristiriidattomasti vastaavat kriittisiä vaatimuksia. Tunnistetaan kriittiset arkkitehtuuriosat ja suunnitteluosuudet sekä niitä vastaavat suunnitteluvirheet eli ne tekijät, jotka eivät noudata spesifiointia. Suunnitteluvaiheessa kiinnitetään huomiota seuraaviin ominaisuuksiin:

- logiikan oikeellisuus
- tiedon oikeellisuus
- rajapintojen oikeellisuus
- rajoitteiden oikeellisuus
- suoritettavuus.

Logiikkasuunnittelun oikeellisuus kohdentuu yhtälöiden, algoritmien ja ohjausten tarkoituksenmukaisuuden todentamiseen. Suunnittelu kohdennetaan vain kriittisiksi todettuihin moduuleihin eli niihin, joiden todetaan sisältävän kriittisiä vaatimuksia tietyllä luotettavuusprofiililla. Kriittisten ja vaarattomien moduulien vuorovaikutus tunnistetaan: kaikki sellaiset komponentit, joiden lähtöjä käytetään kriittisissä komponenteissa, määritetään myös tietynasteisiksi kriittisiksi komponenteiksi. Vastaavasti ohjauslogiikka, jolla herätetään kriittinen komponentti suoritukseen, on kriittisyysvaatimusten alai-

nen. Myös suunnittelun täytyy olla jäljitettävissä vaatimuksiin. Nimenomaan varmistutaan siitä, että kaikki kriittiset vaatimukset sisältyvät suunnitteluun ja että suunnittelu ei sisällä sellaisia uusia ratkaisuja, jotka eivät pohjaudu vaatimuksiin. Loogisessa tarkastelussa suunnittelukuvaukset, ohjausvuot ja yksityiskohtaiset suunnittelut (moduulisuunnittelu) analysoidaan vaatimusten täydellisen ja virheettömän toteuttamisen kannalta.

Tiedon oikeellisuudessa kriittiset tietomäärittelyt, -kuvaukset ja -rakenteet ovat eheitä ja vastaavat vaatimuksia. Tietosuunnitteluun sisältyvät tietoalueiden ja -tarkkuuksien sekä vakioarvojen tarkistukset. Välimuuttujien tarkkuuksien on tuettava tulosten tarkkuusvaatimuksia. Tiedonkäyttö ja käyttöjärjestys arvioidaan siten, että tiedonkäyttö on yhdenmukainen tietorakenteiden, -tarkkuuksien ja -alueen kanssa, ts. loogisuuteen kuuluva tieto tulee myös arvioida. Lisäksi kiinnitetään huomiota tarkoituksettomaan tiedonkäyttöön ja erityisesti kriittisen tiedon asettamiseen ei-kriittisestä logiikasta.

Rajapintojen oikeellisuudessa merkittävintä ovat rajapintasuunnitelmat ei-kriittisten ja ohjelmisto- sekä laitteistorajapintojen välillä. Suunnitteluun kuuluvat parametrien ja tiedon arvioinnit, ohjaussuuntien ja tietotyyppien oikeellisuusarvioinnit.

Rajoitusten oikeellisuudessa merkittävää ovat riskit, jotka johtuvat työkalujen, menetelmien, ohjelmointikielten ja kääntäjien käytöstä. Arvioidaan kaikki reaalimaailman rajoitusten ja suunnitteluratkaisujen aiheuttamat ohjelmistovaatimuksiin kohdistuvat rajoitukset sekä erityisesti se, ettei uusia vaaroja ole suunniteltu.

Suoritettavuuden varmistamiseksi suunnitellaan ajastukset ja mitoitukset ja tarkistetaan, että edellisen eli vaatimusvaiheen tarkistustulokset ovat vielä voimassa. Kielteisessä tapauksessa vaatimusmäärittelyn ajastus- ja mitoitustarkastelu uusitaan.

4.3 Ohjelmiston toteutusvaihe

Toteutus- eli ohjelmointivaihe on ohjelman kirjoitusvaihe virheettömään koodiin asti. Virheettömyyteen ja yhdenmukaisuuteen edellisen vaiheen kanssa kiinnitetään tässä vaiheessa paljon enemmän huomiota kuin edellisissä vaiheissa. Todentamismahdollisuudet parantuvat huomattavasti, jos on olemassa kaksisuuntainen jäljitettävyysmatriisi detaljisuunnittelun ja toteutuksen välillä.

Oikeellisuuden arvioinnissa tunnistetaan kriittiset ohjelmistomoduulit ja tietorakenteet sekä poikkeamat vaatimuksista. Sovelluksesta riippuen tulisi tarkistaa, että koodi, tietorakenteet sekä sisäiset ja ulkoiset rajapinnat vastaavat suunnittelua. Arvioidaan koodin haarautumat, hypyt, luupit sekä keskeytykset. Tietomäärittelyt ja tietojen käyttäminen tarkistetaan ja varmistutaan siitä, että ne vastaavat suunnittelua. Tietotyyppi tarkistetaan. Varmistetaan, ettei tietorakenteen virheellinen käyttäminen aiheuta mahdollista vaaraa.

Liittymistä arvioidaan aliohjelmien kutsut muihin ohjelmistokomponentteihin, yhteisen tiedon kulkeminen sekä viestien kulkeminen tietoliikenteen ja ulkoisten laitteistorajapintojen kautta.

Lisäksi kiinnitetään huomiota ohjelman *suoritettavuuteen* vaatimusten, suunnittelun ja tietokonejärjestelmän asettamissa rajoitteissa, joita ovat fyysiset ja matemaattiset rajoitukset sekä tarkkuuteen, kokoon ja nopeuteen liittyvät rajoitteet. Ajastusten ja mitoitusten suunnittelussa tarkistetaan mahdolliset muuttuneet ominaisuudet.

Logiikka voidaan myös konstruoida koodin pohjalta ja verrata vuokaaviota suunnittelukuvauksiin. Työkalujen, menetelmien, ohjelmointikielten mahdolliset vaarat toteutukselle tulisi arvioida samalla tavalla kuin suunnitteluvaiheessa.

Todentaminen kohdistuu arkkitehtuurin kunkin komponentin (moduulisuunnittelu) toteuttamiseen. Todennusvaatimukset ovat seuraavat:

- Komponenttien toteuttaminen vastaa alemman tason vaatimuksia ja arkkitehtuuria.
- Alemman tason vaatimukset ovat yksiselitteisiä ja ristiriidattomat.
- Jokainen alemman tason vaatimus on jäljitettävissä ylemmän tason vaatimukseen tai vaatimuksiin.
- Suojaavien ja vikasietoisten toimenpiteiden, mekanismien ja algoritmien toteuttaminen on täsmällistä ja johdonmukaista.
- Komponenttien toteutus on todennettavissa.
- Komponenttien toteutus vastaa standardeita.

4.4 Ohjelmiston ylläpito

Vaatimuksia, suunnittelua, koodia sekä laitteisto- ja käyttöohjeita muutetaan ohjelmistotuotannon aikana tai sen jälkeen. Muutoksissa merkittävää on niiden mahdollisten kriittisten vaikutuksien tunnistaminen, mikä tapahtuu tunnistamalla ensin muuttuneet vaatimukset. Vaatimusten luotettavuusprofiili on myös saattanut muuttua samoin kuin vastaavasti sen allokointi ohjelmisto- ja laitteistokomponenteille. Muutokset voivat myös vaikuttaa kohteen käyttöön, käyttöohjeisiin ja analyysi- ja testitoimiin.

Suunnittelu- ja koodimuutoksissa tunnistetaan muutetut kriittiset komponentit ja varmistutaan, että vain tietyt komponentit ovat muuttuneet. Ei-kriittisten komponenttien muutoksissa tarkistetaan etteivät muutokset vaikuta kriittiseen osuuteen.

Käyttöön liittyvissä muutoksissa tarkistetaan käyttöliittymät. Käyttöliittymien käytettävyydellä on kaksi lähes samaa merkitystä. Hyvältä käyttöliittymältä edellytetään oppimishelppoutta ja käyttöön opastettavuutta, mitkä ovat edellytyksiä korkealle luotettavuustekniselle käytettävyydelle (engl. availability) sekä käytettävyydelle (engl. usability) todellisen käyttäjän käyttämänä, todellisissa käyttötilanteissa. Käytettävyydellä on selkeä syy-yhteys myös turvallisuuteen.

Muutokset voivat vaikuttaa luotettavuusprofiilin liialliseen kasvuun, mikä merkitsee kriittisyyden lisääntymistä. Kriittisyyden alentamiseksi pitää mahdollisesti joko muuttaa vaatimuksia tai lisätä luotettavuuden parantamiseen tähtääviä toimenpiteitä ja siten myös vastaavia parantamiseen liittyviä ohjelmistovaatimuksia.

4.5 Tarkistukset

Käsitellään tässä yhteydessä ensisijassa ryhmätöinä suoritettavia tarkistustekniikoita, koska niiden soveltuvuus luotettavuusanalyysien täydennykseksi on merkittävää. Tarkistukset voidaan myös yhdistää eräisiin luotettavuuden analyysitekniikoihin. Yleisimmät tarkistusmenetelmät ovat *Fagan-tarkistukset*, läpikäynnit ja formaalit *suunnittelukatselmukset* sekä *tarkistuslistamenettely*.

Tarkistuksilla tarkoitetaan sellaisia staattisia analyysitekniikoita, joilla todennetaan kohde suorittamatta sitä. Niiden tavoitteena on löytää virheitä määrittelyssä, suunnittelussa tai toteutuksessa tarkastamalla niihin kuuluvaa dokumentaatiota. Tarkistukset ovat ryhmätyötä, joita tehdään valituissa kohdin ohjelmiston kehittämistä. Ne saattavat olla formaaleita tarkastuksia, mm. Fagan-tarkistukset, mutta useimmiten ovat epäformaaleja katselmuksia, joissa harvoin hyödynnetään standardeita tai edes tarkistuslistoja.

Tarkasteluissa tarvitaan kohdetta koskeva dokumentaatio: spesifikaatiot, suunnittelu- ja toteutusdokumentaatiot, verifiointisuunnitelmat, testisuunnitelmat jne. Tarkistukset eivät vaadi erityiskoulutusta; tarkastajat tutustuvat vain sääntöihin, jotka yleensä ovat yritysten itseään varten kehitettyjä.

Tarkistukset ovat laadunvarmistustekniikoista tehokkaimpia seuraavista syistä [Ippolito & Wallace 1995, Bishop 1990]:

- Tarkistukset sopivat kaikkiin ohjelmistotuotannon elinkaaren vaiheisiin Ne tunnistavat todistettavasti virheitä koodista ja dokumentoinnista.
- Tarkistuksilla tunnistetaan ne virhealttiimmat ohjelmaosat, joihin voidaan testein tai lisäanalyysein kiinnittää täsmällistä huomioita

- Tarkistukset ovat hyvin tehokkaita tunnistettaessa loogisia suunnittelu- ja ohjelmointivirheitä.
- Tarkistukset ovat kustannustehokkaita verrattuna siihen, että virheet löydetään integrointi- tai hyväksyntätesteissä.
- Suunnittelijan ja ohjelmoijan taidot parantuvat tarkistuksiin osallistumisella.
- Tarkistuskäytännön kehittymistä on mahdollista seurata tilastoimalla virheentunnistus-kustannussuhdetta.

Tarkastuksiin kohdistuvia haittoja ovat:

- Ohjelmoijat voivat kiusaantua tarkistuksista.
- Tarkistuksia ei yleensä riittävästi oteta huomioon laadittaessa aikataulutusta ja budjetointia.
- Seurantaan, mm. uusiin testeihin, ei olla varauduttu.

Fagan-tarkistusten tavoitteena on tunnistaa virheitä dokumenteista ohjelmiston kehitys-, käyttö- ja ylläpitovaiheissa. Tarkistusprosessi jakaantuu suunnitteluun, valmisteluun, tarkistustilaisuuteen, havaittujen virheiden korjaamiseen ja seurantaan. Kaikilla vaiheilla on selkeät tavoitteet.

Tilaisuuteen osallistuu yleensä 4–6 asiantuntijaa, joista yksi on tarkasteltavasta asiasta riippumaton tarkistustilaisuuden puheenjohtaja ja valmistelija ja yksi tarkasteltavan kohteen vastuuhenkilö. Tilaisuudessa jokainen osallistuja analysoi tarkasteltavan kohteen jostakin tietystä tarkastelukulmasta. Kaikki havaitut virheet kirjataan, mutta niitä ei korjata tilaisuuden aikana.

Tilaisuus etenee seuraavasti: Puheenjohtaja on jakanut etukäteen tarkasteltavan dokumentaation (spesifikaatio, testidata, lähdekoodi, jne.) kaikille osallistujille. Jokaista osallistujaa pyydetään analysoimaan tarkasteltava dokumentaatio, usein tietystä näkökulmasta nähtynä. Yhteistilaisuuksien tarkoituksena on löytää mahdollisimman paljon virheitä, jotka kirjataan mutta joita ei korjata tilaisuuden aikana. Ollakseen tehokas istunto saa kestää vain pari keskeytymätöntä tuntia.

Läpikäynnissä tarkastusryhmä valitsee pienen joukon paperilla olevia testitapauksia, jotka edustavat syötteitä ja näitä vastaavia odotettavissa olevia tulosteita. Tällä testidatalla käydään manuaalisesti läpi ohjelman logiikka esim. Fagan-tarkistuksen tai formaalin suunnittelukatselmuksen mukaisin keinoin. Läpikäynnin suurimpana etuna on se, että sillä tunnistetaan ohjelman virhealttiimmat osat, joita tarkastellaan lähemmin. Sillä löydetään myös tehokkaasti logiikka- ja koodausvirheitä.

Kun Fagan-tarkistuksissa etsitään virheitä, niin *formaaleissa suunnittelukatselmuksissa* tarkastellaan uuden tuotteen sopivuutta käyttötarkoitukseensa. Katselmuksen johtaja asettaa katselmukselle tavoitteet (tarkasteltavat asiat ja odotetut tulokset; kuvaus katselmuksen suorittamisesta ja henkilöiden rooleista, vaadittava yksityiskohtaisuus ja formaalisuus sekä katselmuksen esityslista). Seuraavaksi jaetaan tehtävät ja katselmoidaan esityslistan mukaisesti. Lopuksi raportoidaan.

Läpikäynnit eivät ole niin pakonomaisia kuin formaalit tarkastukset, niissä ei yleensä turvauduta standardeihin tai tarkistuslistoihin. Yleensä niihin ei tarvitse valmistautua.

Tarkistuslistojen käytöllä pyritään kiinnittämään huomiota siihen, että kriittiset seikat on käsitelty kattavasti järjestelmän elinkaaren kaikissa vaiheissa. Yksityiskohtaisia vaatimuksia ei niinkään ole tarkoitus asettaa. Saatavilla olevat tarkistuslistat ovat luonteeltaan yleisiä ja ne joudutaan tulkitsemaan sovelluskohtaisesti. Liian yksityiskohtainen tarkistuslistakäytäntö voi muodostua myös rasitteeksi.

5. Analyysimenetelmät ja -tekniikat

Tässä julkaisussa käytetty käsite *luotettavuuden analyysitekniikka* tunnetaan monilla eri nimillä: riskianalyysin, vaara-analyysin ja turvallisuusanalyysin tekniikat. Niitä ovat vika- ja vaikutusanalyysit, joilla tarkastellaan virhetoimintoja ja niiden seurauksia järjestelmälle ja ympäristölle, sekä vikojen tunnistusanalyysit, joilla etsitään virhetoimintoihin johtavia syitä. Niihin perustuvat tämän julkaisun ohjelmistovaatimusten kelpoistustekniikat.

Vika- ja vaikutusanalyysit ovat saamassa erityistä huomiota ohjelmistoluotettavuuden menetelmien tutkijoiden ja kehittelijöiden keskuudessa. Luotettavuus on ongelma ja pääsyy puheisiin ohjelmistokriisistä. Ohjelmiston luotettavuuden tekniikoita, laskentamalleja, metriikoita, laatujärjestelmiä ym. on tutkittu ja kehitetty jo muutama vuosikymmen, mutta toimivaa menettelyä ei ole saatu aikaan.

Ylhäältä-alas-jakamisperiaate on helposti omaksuttavissa myös ohjelmistoympäristöön, koska nykyisillä ohjelmointitekniikoilla toiminnallisuus esitetään lohkokuvausten avulla. Riskiä on määrällisesti arvioitu lähinnä tunnistamalla merkittävät järjestelmävaatimukset ja luokittamalla ne odotetun vikaantumisen mukaan.

Suurimmat esteet fyysisille laitteistoille tarkoitetun tekniikan soveltamisessa loogiselle ohjelmistolle on ohjelmistovirheiden systemaattinen ja näkymätön luonne. Leveson [1995] on yhdistänyt kaksi loogista periaatetta: vikapuun loogisen komponenttirakenteen ja ohjelman loogisen käskyrakenteen. Koska ohjelma on kuvattavissa vikapuuna, voidaan osoittaa vikapuuanalyysin menettelyllä ja tarkennetuilla formaaleilla todennustekniikoilla, ettei tiettyjä virhetoimintoja voi esiintyä.

Levesonin menetelmä ei perustu kvantitatiivisen arvion tekemiseen, vaan hänen mukaansa, jos epäillään ohjelmavirhettä todennustekniikoiden käytön jälkeenkin, tulisi mieluimmin muuttaa suunnittelua riskittömämpään suuntaan kuin kvantifioimalla yrittää varmistua riittävästä luotettavuudesta. Uudelleen suunnittelulla ei kuitenkaan saada aikaan riskittömyyttä, sillä aina jää jäljelle merkittävän virhetoiminnan mahdollisuus ja siten tietyillä kriittisillä aloilla (lentoliikenne, avaruus, ydinvoima) kvantitatiivinen turvallisuusarviointi on paikallaan. Monilla muilla aloilla, joissa riskit eivät ole suuret, vikapuuanalyysin tyylisellä tekniikalla saadaan kasvatettua luottamusta järjestelmän luotettavaan ja turvalliseen toimintaan, vaikka ei kyettäisikään selvittämään riskittömyyden määrää.

Kvantitatiiviset luotettavuusarvionnit eivät ole vielä valmiita käytännönläheiseen työskentelyyn kaikilla aloilla. Laitteiston arvioinneissa kvantitatiivisilla vaara-analyyseilla on merkittävä asema ja puute ohjelmistoanalyyseissa on suuri menetys.

5.1 Luotettavuuden arvioinnin puitteet

Vaara-analyysien tavoitteena on tunnistaa kahdenlaisia tapahtumia, tapahtumat, joista järjestelmän on suojauduttava; sekä tapahtumat jotka järjestelmän on suojattava. Usein päätöksenteko analyysitulosten toteuttamisesta ei ole helppoa. Siksi päätöksentekoa tehostetaan priorisoimalla tunnistetut vaarat merkittävyyden mukaan.

Mutta mitä ovat vaara-analyysit? Eroavatko ne ohjelmiston staattisista analyyseista ja testeistä, joiden tarkoituksena on tarkasteltavan ohjelmiston oikeellisuuden arviointi tunnistamalla ohjelmistovirheitä ja -virhetoimintoja? Vaara-analyysiprosessi noudattaa yleistä seuraavassa Taulukossa 6 esitettyä riskianalyysiprosessin kaavaa.

Taulukko 6. Riskianalyysin yleinen kulku.

Vaihe	Tehtävä
1	Kohteen määrittely
2	Vaarojen tunnistaminen ja alustava seurausten merkityksen arviointi
3	Riskien suuruuden arviointi
4	Todentaminen
5	Dokumentointi
6	Analyysin päivittäminen

Periaatteessa myös useat staattiset analysaattorit noudattavat samaa kaavaa. Kohde määritellään, ja jollakin järjestelmällisellä tavalla selvitetään, mitä kohteita järjestelmästä tai ohjelmistosta tulisi tarkastella. Kummallakin menetelmällä tarkastellaan siis mahdollisia vaaroja ja etsitään niihin syitä, esimerkiksi syntaktisia tai semanttisia ohjelmiston virhetoimintoja. Tekniikat kuitenkin eroavat. Vaara-analyysit palvelevat ym. kaavan mukaisesti ensisijaisesti mahdollisten vaarojen tunnistamista ja vasta toissijaisesti todellisten vaaroihin johtavien syiden tunnistamista. Staattiset analysaattorit palvelevat ensi sijassa jälkimmäistä eli virheitten tunnistamista ja vaativat erillisen menetelmän niiden tapahtumien ja kohtien tunnistamiseen, mitä analysaattoreilla on tarkoitus tutkia.

Vaara-analyysit ja staattiset analyysit voivat olla päällekkäisiä, toistensa poissulkevia menetelmiä tai täydentäviä, tiettyä tehtävää toteuttavia menetelmiä. Jälkimmäistä vaihtoehtoa noudatetaan tämän julkaisun menetelmäesittelyssä. Luvussa 7 esitettävä menetelmä noudattaa ym. kaavaa. Syiden tarkempi tunnistaminen kuuluu ensi sijassa muille menetelmille, mutta vaara-analyysit luonteensa mukaisesti sopivat kuitenkin tiettyjen syiden tunnistaminen. Niitä ovat etupäässä tietyn luotettavuusattribuutin heikkenemi-

seen liittyvät syyt kaikissa ohjelmiston kehitysprosessin vaiheissa. Seuraavaksi tarkennetaan vaara-analyysien suhdetta muihin todentamisen ja kelpoistuksen menetelmiin.

Ohjelmiston luotettavuuden arviointi koostuu viitteen [Deswarte et al. 1999] mukaan neljästä perusarvioinnista:

1. Luotettavuusvaatimusten kelpoistaminen, minkä tavoitteena on varmistua siitä, että määritellyt luotettavuustavoitteet[2] ja -olettamukset kattavat järjestelmätasojen tunnistetut vaarat ja uhat täydellisesti ja ristiriidattomasti sekä, että ne vastaavat alkuperäisiä tavoitteita.
2. Oikeellisuuden todentaminen, jonka tavoitteena on varmistua siitä, että kehitysprosessi toteuttaa virheettömästi sille asetetut toiminnalliset vaatimukset.
3. Luotettavuuden kelpoistaminen tarkistaa, että luotettavuustavoitteet kattavat tehokkaasti tarkasteltavan kohteen.
4. Prosessin laadun arviointi, mikä perustuu siihen, että jos kehitysprosessi ei noudata hyvin määriteltyä prosessia, tuloksena ei voi olla luotettava järjestelmä.

Luotettavuusvaatimusten kelpoistaminen on em. viitteen mukaan suoritettujen järjestelmätasojen vaara-analyysien täydellisyyden ja ristiriidattomuuden tarkistamista sekä toiminnallisten vaatimusten luotettavuusosuuksien kattavuuden tarkistamista koko kehitysprosessin aikana. Jälkimmäinen edellyttää viitteen esittämän menetelmäkuvauksen mukaan sellaisen luotettavuuslinjan määrittelyä, johon kattavuutta verrataan.

Tässä julkaisussa keskitytään luotettavuusvaatimusten kelpoistamismenetelmiin, ja niitä tarkastellaan täsmällisemmin erillisissä kohdissa 5.2 ja 5.3.

Viitteen esittämä *oikeellisuuden todentaminen* muistuttaa laadun todentamista siten, että tarkastetaan eri prosessivaiheissa vaiheille asetettujen spesifikaatioiden virheetön toteuttaminen tuotosdokumentaatiossa. Oikeellisuuden todentaminen täydentää luotettavuusvaatimusten kelpoistamista: edellinen tarkistaa vaiheelle asetettujen spesifikaatioiden toteutumisen, jälkimmäinen alkuperäisten tavoitteiden toteutumisen jokaisessa vaiheessa.

[2] Luotettavuustavoitteilla tarkoitetaan tässä toiveita ja tarpeita, jotka kohdistetaan ohjelmistoon.

Oikeellisuuden todentaminen koostuu seuraavista toimista, joiden valitseminen tarkasteluun riippuu kohteen monimutkaisuudesta ja kriittisyydestä:

1. Staattiset analyysit, kuten tarkistukset, läpikäynnit ja katselmukset, joita voidaan soveltaa kaikissa kehitysprosessin vaiheissa jokaiselle vaihedokumentille (spesifikaatiot ja suunnitteluaineisto, todennussuunnitelmat, jne.).
2. Testit, jotka sisältävät useita tekniikoita, kuten toiminnalliset, rakenteelliset, raja-arvotestit, jne. Testitekniikat kattavat erityisesti ne järjestelmäkäyttäytymisen dynaamiset ominaisuudet, joita on vaikea tunnistaa staattisin analyysein tai formaalein menetelmin.
3. Formaalit menetelmät ja oikeaksitodistamiset, joista edelliset hyödyntävät selkeiden, täsmällisten ja yksiselitteisten vaatimusten, olettamusten, spesifikaatioiden ja sunnitteluratkaisuiden kuvaamista. Yhdessä jälkimmäisten toimien kanssa ne pyrkivät osoittamaan toteutuksen vastaavuuden sille asetettuihin ominaisuuksiin.
4. Käyttäytymisanalyysit perustuvat joko spesifikaatiosta, suunnittelusta tai toteutuksesta matemaattisesti johdettuun kohteen käyttäytymismalliin. Mallien avulla todennetaan yleisiä ominaisuuksia, kuten täydellisyyttä ja yhtenäisyyttä, sekä tiettyjä vaatimuksista johdettuja ominaisuuksia, kuten tietyn tapahtumasekvenssin olemassaoloa. Käyttäytymismallit ovat päätöstaulukoita, Petriverkkoja, tilakoneita, jne.
5. Jäljitettävyysanalyysit koostuvat ristiviittauksista ja matriiseista. Ne palvelevat kolmea päämäärää:
   - jäljittämistä vaiheelle asetuista vaatimuksista toteutukselle
   - jäljittämistä toteutuksesta vaatimuksille
   - jäljittämistä vaatimuksista testitapauksiin ja todennustoimiin.

Staattisin analyysein voidaan valtaosa virheistä selvittää heti niiden syntymishetkellä. Toimet ovat helppokäyttöisiä ja eivät kovin kalliita, niiden tehokkuus on hyvä mutta ne edellyttävät kuitenkin tarkkaa paneutumista toimiin.

Riippumatta testivaiheesta (moduuli-, integrointi- tai järjestelmätestaus) eri testitekniikoita hyödynnetään rinnan. Tilastolliset todennäköisyystestit täydentävät deterministisiä testejä silloin, kun testattavien tapausten määrä on erittäin suuri. Tilastoilla kohdennetaan determinististä testaamista.

Luotettavuuden kelpoistusmenetelmät koostuvat seuraavista toimista, joista kaksi ensimmäistä kuuluvat attribuuttiin tietoturva ja kolmas on yleinen soveltuen attribuutteihin:

1. Tunkeutumisanalyysi, jolla järjestelmällisesti tunnistetaan virheitä, jotka voivat johtaa ei-toivottuihin tilanteisiin.
2. Salapolkuanalyysi, jolla etsitään tietoturvan luottamuksellisuusominaisuuden heikkouksia tai dokumentoimattomia tahattomasti tai tahallisesti piilotettuja sisäisiä tai ulkoisia takaportteja ohjelmistojärjestelmän tietoihin.
3. Eksperimentaalinen arviointi, jossa tarkastellaan järjestelmän kriittisimpien osien luotettavaa käyttäytymistä. Arviointi perustuu todellisessa käyttöympäristössä kerättyyn luotettavuustietoon. Tekniikoita ovat virheen syöttäminen ja tilastollinen merkittävyyden arviointi.

Prosessilaadun arvioinnin tekniikoita ovat tarkistukset, katselmukset, auditoinnit, jne. Niillä pyritään selvittämään, että projekti on asianmukaisesti suunniteltu ja ohjattu. Arvioimiseksi on kehitetty laatujärjestelmiä, mm. Capability Maturity Model ja ISO 9000, jotka osaltaan keskittyvät virheitä ennaltaehkäisevään hallintaan.

5.2 Luotettavuusvaatimusten kelpoistaminen

Luotettavuusvaatimusten kelpoistamismenetelmillä tarkoitetaan menetelmiä, joiden perustehtävänä on todentaa ohjelmistotuotannon toteutuminen alkuperäisiin luotettavuusvaatimuksiin. Luotettavuusvaatimusten kelpoistaminen koostuu seuraavista toimista:

- alustava vaara-analyysi
- vaara-analyysi
- yhteisvika-analyysi
- kvantitatiivinen luotettavuusarviointi.

Kolmen ensimmäisen toimen tiedetään hyödyntävän kaikkia luotettavuusattribuutteja, mutta viimeinen, kvantitatiivinen luotettavuusarviointi, on vielä kehittymässä. Sen merkitys ohjelmiston luotettavuuden arvioinnissa keskittyy vain kaikkein kriittisimpiin sovelluksiin.

5.2.1 Alustava vaara-analyysi

Alustava vaara-analyysi (Preliminary Hazard Analysis, PHA) on ylimmän tason kvalitatiivinen menetelmä ohjelmiston kehityksen elinkaaren varhaisissa vaiheissa. Sillä tun-

nistetaan kriittisiä vaaratekijöitä ja määritellään järjestelmän luotettavuusvaatimukset. Tarkastelu perustuu erilaisiin läpikäyntityylisiin tarkasteluihin ja tarkistuslistoihin. Tuloksena luokitellaan alustavasti järjestelmän ohjelmistosta tai laitteistosta aiheutuvat riskit ja päätetään lisätarkastelujen kohdistamisesta mm. kohdistamalla luotettavuutta parantavia järjestelyjä suunnittelulla. Menetelmä on myös mahdollista tehdä poikkeamatarkastelun (HAZOP) tai vika- ja vaikutusanalyysin (FMEA) tyylisenä tarkasteluna. Tulokset dokumentoidaan ja dokumentti liitetään mukaan suunnitteluaineistoon, jotta suunnittelussa voidaan arvioida erilaisten vaihtoehtojen tärkeyttä.

PHA:n paras soveltamisaika on silloin, kun järjestelmän määrittelyvaiheessa ohjelmiston tehtävä on määritelty, mutta myös esisuunnitteluvaiheeseen sitä on käytetty. Menetelmään liitetään usein ohjelmistovaatimusten turvallisuuden analysoiminen.

Menetelmän merkitys luotettavuuden analysoimisessa on siinä, että sen pohjalta voidaan priorisoida merkittävimmät kriittiset kohdat, funktiot, laitteet tai ohjelmistot, joiden jatkotarkasteluun kannattaa kohdistaa lisäponnistuksia. Tuloksena on myös suositus sopiviksi analysointitekniikoiksi.

5.2.2 Vaara-analyysi

Ohjelmiston vaara-analyysit ovat joukko peräkkäisiä ja iteroivia kelpoistustoimia järjestelmän allokointitasoilla (toiminto, osatoiminto, osajärjestelmä, komponentti, jne.). Jokaisella tasolla tavoitteena on tunnistaa potentiaalisia kyseisen tason luotettavuusvaatimuksiin vaikuttavia virhetoimintoja ja vaaroja sekä niihin johtavia syitä. Kaikissa vaara-analyyseissa tarkastellaan virhetoimintojen merkittävyyttä luotettavuustavoitteiden kannalta sekä määrätään virhetoiminnon tai vaaran kriittisyys. Kaikki vaara-analyysit suositellaan tehtävän perinteisillä riskianalyysin tekniikoilla.

Ohjelmistovaatimusten vaara-analyysi (Software Requirements Hazard Analysis, SRHA) perustuu PHA:n ja mahdollisen esisuunnittelun tuloksiin, ja menetelmä on PHA:n jatkoa tunnistettaessa kriittisiä ohjelmiston spesifikaatiovirheitä. Kriittisyyden määrää riski, joka liittyy spesifikaatiovirheisiin. SRHA:ssa tarkastellaan järjestelmätason vaatimuksia, rajapintadokumentteja ja ohjelmiston vaatimusmäärittelyä.

SRHA soveltuu hyvin suunnittelun varhaiseen ohjaukseen, jossa

- tunnistetaan kriittiset ohjelmistovaatimukset
- varmistetaan, että kriittiset vaatimukset ovat oikeita ja täydellisiä
- suositellaan kriittisten kohtien parantamista tai testitapauksia.

SRHAlla tunnistetaan kriittisiä kohteita, joita järjestelmätason PHA ei ole kattanut. Menetelmä soveltuu sellaisten spesifistisen vaatimusten tarkasteluun, kuten raja-arvot, ajastukset, äänestyslogiikka, kriittisten käskyjen prosessointivaatimukset, kytkentälogiikka jne. Tuloksia hyödynnetään myöhempien ohjelmiston elinkaarivaiheiden vaaraanalyyseissa, kuten mm. moduulisuunnittelun ja koodin analyyseissa. Tulokset auditoidaan laatujärjestelmän mukaisesti esimerkiksi ohjelmiston vaatimusmäärittelyn katselmuksissa.

SRHA:ssa analysoidaan tai katselmoidaan systeemi-, osasysteemi- ja liitäntävaatimusspesifikaatioita, ja muita systeemidokumentteja tarkoituksena selvittää, että 1) kriittiset vaatimukset on oikein allokoitu ohjelmistolle, 2) kriittiset kohteet PHA:n tuloksista on tunnistettu ja 3) kriittisten vaatimusten jäljitettävyys systeemispesifikaatiosta detaljitason ohjelmistovaatimusten spesifikaatioon on olemassa. Lisäksi tarkastelukohteina ovat toimintakaaviot, tietovuokaaviot, ajastuskaaviot ja muu ohjelmistodokumentaatio.

Suurimpana virhelähteenä ohjelmoitavassa järjestelmässä pidetään siirtymistä vaatimusmäärittelystä ohjelmiston spesifikaatioon. Tätä varten spesifikaatiot tulisi kuvata mahdollisimman formaalisti, jolloin tarkastelu tulee mahdolliseksi.

5.2.3 Yhteisvika-analyysi

Järjestelmän arkkitehtuuri voi koostua redundanttisista osista, yhteisistä tai samantyyppisistä komponenteista tai samasta ohjelmistosta. Kriittisen yhteisvirheen eli yksittäisvian vaikutuksesta aiheutuvan usean komponentin kriittisen vikaantumisen mahdollisuus kasvaa näissä tapauksissa. Vikoja, jotka vahingoittavat varmistuksia tai riippumattomia olettamuksia systeemin toiminnasta, kutsutaan yhteisvioiksi.

Yhteisvika-analyysi kuuluu tässä julkaisussa noudatettavan luokitustavan mukaan menetelmiin eikä tekniikoihin lähinnä siksi, että sille ei ole olemassa erillistä suoritustekniikkaa, vaan mm. kaikki perustekniikat FMECA ja FTA soveltuvat tietynlaiseen yhteisvikojen tarkasteluun. Yhteisvika-analyysi voidaan rinnastaa myös muihin vaaraanalyysimenetelmiin siksi, että yhteisvirheet ja -viat ovat erityisen tärkeitä juuri ohjelmistoille, joiden kahdentaminen ei vähennä mahdollisten virhetoimintojen esiintymistä.

Menetelmällä tarkastellaan kohteen yhteisvikoja eri tavoin riippuen yhteisvian vaikutusalueesta. Sisäiset tapahtumavirrat katkaistaan erottamalla yhteisvirheisiin alttiit systeemiosuudet toistaan. Systeemin osat sekä kehitetään että ylläpidetään toisistaan riippumattomasti. Erityisen tärkeää on tunnistaa systeemin osien väliset liitynnät ja vuorovaikutukset. Mahdollisten vikojen vaikutukset systeemin muihin osiin on tutkittava jollakin sopivalla tekniikalla, kuten FMECA:lla.

Edellä kuvattu tapa ei sovellu systeemin ulkopuolelta vaikuttaviin yhteisvikoihin, jotka vaikuttavat systeemin useaan osaan yhteisesti. Näitä riskitapahtumia ovat tulipalo, luonnonilmiöt, säteily, virtakatkot, tietoverkkokatkot jne.

Kolmas tapa on etsiä sellaisia yhteisiä riippuvuuksia kriittisistä ohjelmisto-osuuksista, komponenteista ja laitteistoista, jotka aiheuttavat järjestelmätason vikaantumisen. Näitä ovat mm. yhteiset komponenttityypit, valmistustekniikat, ylläpitoprosessit ja -henkilöt sekä arviointimenettelyt.

5.3 Luotettavuusvaatimusten kelpoistustekniikat

Ippolito & Wallace [1995] ovat koonneet kirjallisuuskatsauksen luotettavuuden tarkastelutekniikoiden eduista ja haitoista. Tässä kohdassa tarkastellaan seuraavien tekniikoiden soveltuvuutta:

1. ohjelmiston vika-, vaikutus- ja kriittisyysanalyysi
2. ohjelmiston vikapuuanalyysi
3. poikkeamatarkastelu
4. tapahtumapuuanalyysi
5. ohjelmiston oikopolkuanalyysi
6. Petriverkot.

5.3.1 Ohjelmiston vika-, vaikutus- ja kriittisyysanalyysi

Ohjelmiston vika-, vaikutus- ja kriittisyysanalyysi (Software Failure Modes, Effects and Criticality Analysis, ohjelmiston FMECA) perustuu laitteiston vastaavaan analyysiin FMECA [IEC 60812 1985]. FMECA on induktiivinen vaara-analyysitekniikka, jolla selvitetään kohteen eri osien vioittumistavat sekä määritellään näiden vaikutukset ja kriittisyydet kohteen muihin osiin sekä kohteelta vaadittuun toimintaan. FMECA osoittaa, miten kohde käyttäytyy erilaisissa vikatilanteissa, miten se antaa tietoa kohteen kyvystä suoriutua tehtävästään ja miten sen tulosten perusteella voidaan tehdä parannuksia, joilla muun muassa vähennetään hallitsevia yksittäisvirheitä ja paljastetaan piileviä virhetilanteita.

FMECA-tekniikalla tunnistetaan järjestelmän tai valmistusprosessin virhetilanteiden vaikutukset; systeemiin sovellettuna sitä kutsutaan järjestelmä-FMECA:ksi ja vastaavasti prosessiin sovellettuna prosessi-FMECA:ksi. Järjestelmä-FMECA voi kohdentua laitteistoon, varusohjelmistoihin, ohjelmistoihin tai inhimillisiin tekijöihin. Nämä koh-

dealueet käsitellään erikseen erillisillä analyyseilla, jotka yhdistetään. Esimerksi järjestelmä-FMECA jakaantuu ohjelmisto-FMECA:han ja laitteisto-FMECA:han, jotka jollakin sopivalla tavalla yhdistetään.

Tekniikkaa on käytetty erityisesti elektroniikkateollisuudessa. Viitteen [Ippolito & Wallace 1995] mukaan se soveltuu myös kaikkien ohjelmiston elinkaaren vaihetuotteitten tarkasteluun vaatimusmäärittelystä koodiin. Lähdekielisestä koodista analysoijat listaavat kohteen kaikkien ohjelmistokomponenttien mahdolliset virheet ja tunnistavat potentiaaliset seuraukset järjestelmälle. Kuitenkaan koodianalyysien suorittamisesta ohjelmiston FMECA:lla ei kirjallisuudessa ole paljon näyttöä, eikä tiedetä mahdollisten tarkasteluiden onnistumisista. Koodissa ollaan syvällä analysoinnin kannalta yksityiskohtaisella tasolla, josta ylöspäin selvittäminen on monimutkaista ja vaativaa.

Ohjelmiston FMECA voidaan aloittaa, kun tarkasteltavasta elinkaarivaiheen dokumentaatiosta on julkaistu tarkastelukelpoista aineistoa (mm. järjestelmäkuvaus tai järjestelmän tai ohjelmiston vaatimusmäärittely). Menetelmän alhaalta-ylös-lähestymistapa voidaan liittää moneen muuhun menetelmään, erityisesti ohjelmiston tai järjestelmän vikapuuanalyysiin. Virheiden tunnistustekniikoista usein poikkeamatarkastelu, HAZOP, on vaihtoehtoinen tarkastelumuoto. HAZOP-tekniikkaa käytetään yleisesti teollisuusprosessien häiriöiden tarkasteluun, kun taas FMECA on yleinen laitteistotasolla. HSIA (Hardware Software Interaction Analysis) muistuttaa tarkastelutapana niin läheisesti FMECA:ta, että tekniikat ovat yhdistettävissä yhdeksi menetelmäksi.

Taulukko 7. Esimerkki SFMECA:n tarkasteluraportin formaatista. Vioittumistapa on kaikille FMEA-tyylisille tarkasteluille keskeisintä.

Tunnisteluku	**Toiminto**	**Vioittumistapa**	**Vioittumistavan mahdolliset vaikutukset:** **a) Toiminto** **b) Muu toiminto** **c) Ympäristö**	**Vioittumistavan mahdolliset syyt:** **a) Laitteisto** **b) Ohjelmisto**	**Kriittisyysaste**	**Huomautukset ja suositukset:** **a) Havaintomekanismi** **b) Riskin ohjaus** **c) Muu**

Taulukko 7 esittää esimerkkiformaattia, johon sisältyvät merkittävimmät ohjelmiston vika-, vaikutus- ja kriittisyysanalyysissa tarvittavat tiedot:

1. Tunnisteluku on järjestysnumero vioittumistavoittain, jotka luetellaan sarakkeessa kolme.

2. Toiminto. Tarkasteltavan kohteen tunniste, tässä ohjelmiston toiminto. Voi olla myös komponentti. Joko toiminto kirjoitetaan sellaisenaan tai viitataan tunnisteeseen tarkasteltavassa dokumentissa.

3. Vioittumistapa. Jokaiselle toiminnolle (sarake 2) laaditaan joukko mahdollisia vioittumistapoja (ohjelmiston osalta mahdolliset virhetoimintotyypit). Luvussa 7 esitetään avainlauseita, joiden avulla virhetoimintotyypit tunnistetaan.

4. Vioittumistavan mahdolliset vaikutukset: a) toiminto, b) muu toiminto, c) ympäristö. Tarkastellaan jokaisen vioittumistavan mahdollisia vaikutuksia ja kirjataan ne sarakkeeseen neljä. Tarkastelussa voidaan hyödyntää muita induktiivisia tekniikoita, kuten tapahtumapuuanalyysia. Vaikutukset voidaan esittää asteittain kyseiselle seuraustasolle. Tasoiksi voidaan valita joko kyseinen tarkasteltava ohjelmistotoiminto tai muu ohjelmistotoiminto, käyttöliittymä, osajärjestelmä tai järjestelmä tai muu järjestelmä tai ihminen, ympäristö tai tehtävä. Myös vaikutuksen suuruutta esittävä sarake voidaan asettaa.

5. Vioittumistavan mahdolliset syyt. Vioittumistapaan johtavat pääsyyt yleisesti esitettynä: laitteistovika, ohjelmistovirhe. Tarkempi syiden tunnistaminen myöhemmin, kun kriittisyydestä ja jatkosta on päätetty. Syiden hakuun voidaan käyttää apuna jotakin deduktiivista tekniikkaa, esimerkiksi vikapuuanalyysia. Haetun mahdollisen virhetilan tai virheen esiintymistiheys voidaan arvioida ja asettaa sitä varten erillinen taulukkosarake.

6. Kriittisyysaste. Luokitellaan kriittisyysaste tiettyyn etukäteen määrättyyn tasoon. Määrittelyssä otetaan huomioon vioittumistavan vaikutukset ja esiintymistiheys, joka arvioidaan vioittumistavan syiden esiintymistodennäköisyydestä. Voidaan myös hyödyntää nk. riskin prioriteettilukumenettelyä (Risk Priority Number, RPN), johon vaikutustason ja esiintymistiheyden lisäksi on kolmantena tekijänä havaittavuus. Havaittavuudelle voidaan asettaa erillinen sarake; kuvan esimerkkitapauksessa se kuvataan sarakkeessa seitsemän. Menettelyssä

tekijät on numeroitu (yleensä 1–10) ja RPN:n saamiseksi ne kerrotaan keskenään. Myös riskitasoa voidaan käyttää kriittisyysasteeksi.

7. Huomautukset ja suositukset: a) havainto b) riskin ohjaus c) muu. Huomautetaan tai suositellaan mistä tahansa asianmukaisesta tiedosta, joita voivat olla vioittumistavan havainto- tai estomekanismi, testaus, jatkotyöt ja -tarkastelut.

Havainnointitavan tulisi olla sellainen, mistä vioittumistavan esiintyminen voidaan käytännössä havaita. Niitä vioittumistapoja tai virhetoimintoja, joille ei analysoinnissa löydy havaitsemiskeinoja, tulisi tuloksissa selkeästi korostaa. Tulosten arviointi voidaan suorittaa usealla tasolla kriittisyydestä, laajuudesta, kompleksisuudesta ja projektoinnin eri asioista. Kriittisimpiin ja yleensä kaikkiin korostettuihin tuloksiin tulisi tehdä aina arviointi ja viedä tulokset parantaviksi toimenpiteiksi esimerkiksi joko uudelleen suunnittelulla tai käyttötoimenpitein. Suositukset näistä annetaan joko analysoinnin tuloksina tai erillisen arviointityöryhmän tuloksina.

Tuotteen tai suunnittelun kompleksisuus ja suunnittelutiedon saatavuus määrittää pitkälti sen, millä tarkkuudella FMECA tulisi tehdä. Yleensä tekotapoja on kaksi: 1) fyysiseen tai 2) toiminnalliseen rakenteeseen pohjautuva. Fyysistä lähestymistapaa kannattaa käyttää silloin, kun kohteesta on riittävästi yksityiskohtaista tietoa. Tämä yleensä merkitsee jo pitkälle vietyä systeemin kehittämistä, ei esimerkiksi prosessin ensimmäisiä osuuksia tai spiraalivaiheita. Toiminnallista lähestymistapaa kannattaa käyttää niissä tapauksissa, joissa systeemi on kompleksinen, yksityiskohtainen fyysinen käsittely vie liikaa aikaa tai kun systeemin fyysinen rakenne ei yksinkertaisesti ole riittävästi tunnistettavissa fyysistä tapaa varten.

Kummallakin lähestymistavalla tulisi tunnistaa tarkasteltavan kohteen tehtävät (esim. ohjattava kohde: kemiallinen prosessi, säätöventtiili), tehtävien mahdollinen vaiheistus sekä käyttötavat. Ympäristöprofiili (-kuvaus) tulisi määrittää tehtävä- ja tehtävävaihekohtaisesti. Jos profiileita on useampia, määritys tulee tehdä kaikille samalla tavalla.

Usein, etenkin kun kyseessä on laaja, kompleksinen tai kriittinen tarkastelun kohde, laaditaan useita luotettavuuslohkokaavioita kuvaamaan kohdetta. Niistä on erityistä hyötyä FMECA-tarkasteluissa.

Taulukko 8 esittää joitakin ohjelmiston FMECA-tekniikan hyötyjä ja haittoja. Erityisesti voidaan todeta, että normaali taulukointiohjelma (esim. Excel) soveltuu hyvin analyysipohjaksi. Erityisesti ohjelmistolle tarkoitettua kaupallista työkalua ei ole saatavilla, mutta automaattisten FMECA-työkalujen käyttö on lisääntynyt voimakkaasti viime vuosina [Ormsby et al. 1991, Bell et al. 1992, O'Rorke 1992, Price et al. 1995,

Montgomery et al. 1996] elektroniikalle. Automatisoinnin avulla aikaansaadan normaalia täydellisemmät ja yhtenäiset FMEA-tuloskaavakkeet halutussa ajassa. Lisäksi eräs merkittävä etu on se, että automaattisella menetelmällä seurataan tuotteen kehitystä koko suunnittelujakson, arkkitehtuurin, osajärjestelmien ja komponenttien, läpi.

Taulukko 8. Ohjelmiston vika-, vaikutus- ja kriittisyysanalyysin edut ja haitat.

Edut	Haitat
Tekniikan perusteet on helppo oppia.	Menettelyyn sisältyvä aivoriihityöskentely on aikaavievä, hidas, raskas ja virhealtis.
Helppo päästä alkuun, taidot kehittyvät tarkastelussa.	Usein aloittelijoille ikävä yksityiskohtaisen tarkastelutavan takia.
Paljastaa odottamattomia vaaroja. Voidaan parantaa luotettavuutta.	Ei käsittele moninkertaisia vikoja.
Soveltuu ohjelmiston elinkaaren kaikkiin vaiheisiin määrittelystä koodaukseen.	Ei ole käytännöllinen sovellettaessa suoraan suunnitteluun ja toteutukseen ilman ennakoivia analyyseja aikaisemmissa vaiheissa.
Järjestelmällinen, mistä on hyötyä luotettavuuden kattavassa osoittamisessa.	Järjestelmällisyydestä voi olla vaikea päästä eroon, mikä vaikuttaa kustannustehokkuuteen.
Perustekniikka, joka on helposti liitettävissä moneen muuhun tekniikkaan.	Kvalitatiivinen tekniikka, mutta muunnettavissa vikapuuksi ja siten kvantifioitavissa.
Runsaasti saatavilla hyviä kaupallisia työkaluja, mutta usein tavallinen taulukointi on riittävä.	Kaupalliset työkalut ovat suhteellisen kalliita monipuolisuutensa tähden. Ohjelmistolle ei ole erikseen työkalua.

Laitteiston ja ohjelmiston rajapinta-analyysi

Laitteiston ja ohjelmiston rajapinta-analyysi (Hardware Software Interface Analysis, HSIA) on liitettävissä FMEA-tyyppisiin analyyseihin tarkastelemalla rajapintoja kahdesta näkökulmasta:

1. Laitteiston vikaantumisten seurauksena joko ohjelmiston suoritus on virheellistä tai puuttuu kokonaan.
2. Ohjelmiston virhetoiminta vaikuttaa laitteiston toimintaan.

Esimerkkejä jälkimmäisestä näkökulmista ovat mm. käskyt, jotka joko a) puuttuvat kokonaan tai tulevat b) liian aikaisin, c) liian myöhään tai d) virheellisinä.

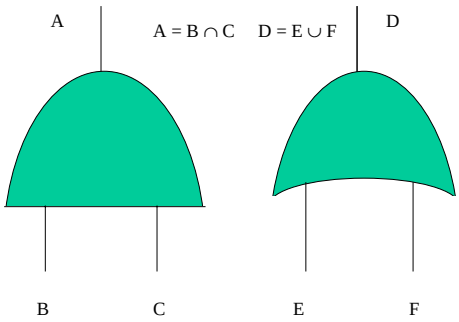
5.3.2 Ohjelmiston vikapuuanalyysi

Bell Telephone Laboratoriesin H. Watson kehitti vuonna 1961 vikapuuanalyysin Yhdysvaltojen ilmavoimille toimintavarmuuden ja turvallisuuden tarkasteluun. Vikapuuanalyysilla etsitään tiettyihin vaaratapahtumiin – huipputapahtumiin – johtavia syitä. Analyysi on deduktiivinen tekniikka, jolla ei tunnisteta vaaroja, vaan huipputapahtumien täytyy olla tiedossa ennen tarkasteluun ryhtymistä. Ne täytyy valita huolellisesti, sillä liian geneerinen huipputapahtuma johtaa laajaan vikapuuhun ja liian spesifistinen ei kata kaikkia huipputapahtumaan johtavia syitä. Tavoitteena voi olla tarkasteltavan kohteen

- luotettavuusmallin laadinta
- heikkojen kohtien tai merkittävien vikayhdistelmien tunnistaminen
- valvontojen ja varmennusten riittävä arviointi
- luotettavuuden kvantitatiivinen määrittäminen.

Vikapuu on looginen kaavio, joka esittää kohteen kriittisen vikaantumisen eli huipputapahtuman riippuvuuden kohteen osien vioista tai ulkoisista tapahtumista. Analyysissa lähdetään liikkeelle tarkasteltavan kohteen vioittumistapahtumasta selvittämällä, mistä tapahtumasta tai tapahtumakombinaatiosta se voi aiheutua. Edetään kohti yksinkertaisia, toisistaan riippumattomia perustapahtumia, joiden esiintymisestä on saatavissa kokemusperäistä tietoa. Löydetyt syyt: viat ja virheet joko korjataan tai esitetään sellaisia ehkäiseviä, suojaavia tai muita toimenpiteitä joilla tapahtumaketjut voidaan pysäyttää. Viat ja virheet voivat aiheutua ohjelmoinnista, suunnittelusta, inhimillisistä tekijöistä tai laitteistosta.

Analyysin tulokset esitetään graafisena kaaviona, jossa tapahtumien eri kombinaatiot yhdistetään. Tosin tekstuaalinen muotokin on mahdollista. Kaaviomuodossa tapahtumakombinaatiot kuvataan porteilla, jotka sallivat tai estävät tapahtuman etenemisen tiettyyn seuraustapahtumaan. Portteja on useita tyyppejä, joista kaksi on yleisintä: TAI-portti ja JA-portti (Kuva 6). TAI-portti sallii kaikkien syötetapahtumien etenemisen, JA-portti vaatii kaikkien tapahtumien olevan tosia, ennen kuin seuraustapahtuma voi olla tosi Boolen algebran mukaan. Portit ja tapahtumat yhdistämällä saadaan konstruoitua vikapuu tietylle huipputapahtumalle.



Kuva 6. Vikapuun kaksi yleisintä porttia: JA-portti ja TAI-portti. JA-portissa kaikkien syötetapahtumien pitää olla tosia (Boolen algebra) ennen kuin tulostapahtuma on tosi. TAI-portissa riittää, kun jokin syötetapahtuma on tosi. Bayesin todennäköisyyslaskennassa saadaan arvot P(A) = P(B) x P(C) JA-portille ja P(D) = P(E)+P(F)- P(E) x P(F) TAI-portille.

Vikapuuanalyysi on alun perin tarkoitettu järjestelmä- ja laitteistoanalyyseihin kvantitatiivisen todennäköisyysarvioinnin tekemiseksi [IEC 61025 1990], mutta sitä on menestyksellä sovellettu myös ohjelmistoihin (Software Fault Tree Analysis, ohjelmiston FTA) [Leveson & Harvey 1983, 1991]. Ohjelmiston FTA soveltuu kaikkiin ohjelmistoprosessin vaiheisiin vaatimusmäärittelystä koodausvaiheeseen. Erityisesti Leveson [1991] on soveltanut tekniikkaa järjestelmätasosta koodiin. Sitä sovelletaan yksinomaan kvalitatiivisesti. Kvalitatiivisissa analyyseissa jo pelkkä puun piirtäminen tukee suunnittelijoita luotettavuusratkaisuista päätettäessä.

Ohjelmiston FTA eroaa normaaleista ohjelmiston tarkistustekniikoista siinä, että se pakottaa analysoijan tarkastamaan ohjelmistoa erilaisesta näkökulmasta, kuin mihin hän on tottunut. Analyysi on perustekniikka, jonka ylhäältä-alas-lähestymistapa voidaan liittää tehokkaasti moneen muuhun menetelmään. Tekniikka voidaan liittää myös järjestelmän tai laitteiston (elektroniikan) vikapuuhun. Tällöin laitteiston ja ohjelmiston vikapuut yhdistetään koko järjestelmän analysoimiseksi. Tämä on merkittävää siksi, että monet vaarat voivat aiheutua ohjelmistovirheen, laitteistovian tai inhimillisen virheen yhdistelmästä.

Koska vikapuu kuvaa tapahtumat loogisesti, kaikki logiikkaa noudattavat tapahtumasarjat voidaan kuvata. Myös FMECA voidaan muuntaa vikapuiksi ja arvioida kvantitatiivisesti tietyn tapahtuman luotettavuus. FMECA:lla etsitään usein ne kriittiset tapahtumat, joita tarkemmin tutkitaan vikapuulla. Myös päinvastaista menettelyä on käytetty [Maskuniitty & Pulkkinen 1995]. Tällöin FTA:lla on ensin tunnistettu lyhyet minimikatkosjoukot, joita on tarkemmin tutkittu FMEA:lla. Taulukko 9 esittää ohjelmiston FMECA-tekniikan olennaiset edut ja haitat. Tarkastelu pohjautuu Levesonin töihin [1983, 1991, 1995] ja viitteeseen [Bishop 1990].

Taulukko 9. Ohjelmiston vikapuuanalyysin edut ja haitat.

Edut	Haitat
Voidaan laskea vaaratapahtuman esiintymistodennäköisyys.	Tekniikka on periaatteessa yksinkertainen, mutta todellinen hyöty vaatii kyvykkyyttä, joka saadaan vasta kokemusten avulla.
Paljastaa moninkertaiset eri järjestelmän osissa syntyneet virheet.	Vaikea analysoida kovin laajaa tai kompleksista järjestelmää.
Voidaan tarkastella koko järjestelmää (ohjelmistoa, laitteistoa ja käyttöliittymää).	Vikapuut kasvavat helposti yli käsittelykyvyn.
Vikapuuhun taltioitavaa informaatiota käytetään mm. suunnitteluun ja testaukseen.	Ei ole käytännöllinen järjestelmille, joissa on lukuisia kriittisiä vikoja.
Käytöllä on pitkät perinteet laitteistojen tarkastelussa.	Ajastustekijät vaikeita liittää vikapuihin.
Selvittää virheettömän tilan muuttumisen vaaralliseksi.	On mahdollista osoittaa, että ohjelma ei koskaan saavuta vaarallista tilaa, mutta ei osoittaa virheellisiä mutta turvallisia tiloja.
Koko vikapuun ei tarvitse olla täydellinen analyysin ollessa silti hyödyllinen.	Vikapuun huipputapahtuma kannattaa valita huolellisesti.
Systemaattinen tunnistusmenetelmä.	Kaupalliset työkalut ovat suhteellisen kalliita, koska niihin aina sisältyy myös numeerisen luotettavuuden laskenta.
Saatavilla runsaasti työkaluja.	
Vikapuuta ei tarvitse välttämättä kuvata kaaviomuotoon, vaan usein tekstuaalinen esitystapa on riittävä.	
Perustekniikka, joka on liitettävissä monen tekniikan yhteyteen.	

5.3.3 Poikkeamatarkastelu

Poikkeamatarkastelu (Hazard and Operability Study, HAZOP) on alun perin tarkoitettu kemian laitoksille riskien tunnistusmenetelmäksi, jota käytetään erityisesti toimintaparametrien muutoksista tai puutteellisesta suunnittelusta johtuvien häiriöiden ja niiden aiheuttamien onnettomuusriskien kartoittamiseen [Kletz 1986]. Tekniikka kelpaa myös pienempien vikaantumisten syiden hakuun, ja siten sillä voidaan ainakin periaatteessa kartoittaa ohjelmistosta ohjattavalle kohteelle aiheutuvia ongelmia.

HAZOP suoritetaan käyttämällä avainsanoja (ei, enemmän, takaisinpäin, jne.). Näistä avainsanoista päätellään, minkälaisia kriittisiä seurauksia tai minkälaisia käytön aikaisia ongelmia voi esiintyä. Myös syitä tarkastellaan, mutta vain lähinnä kelpoistettaessa määrätyn avainsanan merkittävyyttä. Suositellaan systeemiä tai prosessia parantavia ratkaisuja. Menetelmä on hyvin suosittu prosessiteollisuudessa tehokkaana työkaluna prosessin riskien kartoituksessa ja vähentämisessä.

Poikkeamatarkastelu on ryhmätyötä, mitä ohjelmiston FMECA ei välttämättä ole. Tarkastelua johtaa koulutettu tekniikan tunteva spesialisti. Istuntoihin osallistuu tarkasteltavan kohteen tuntevia asiantuntijoita. Taulukko 10 esittää keskeisimmät poikkeamatarkastelun hyödyt ja haitat.

Taulukko 10. Poikkeamatarkastelun edut ja haitat.

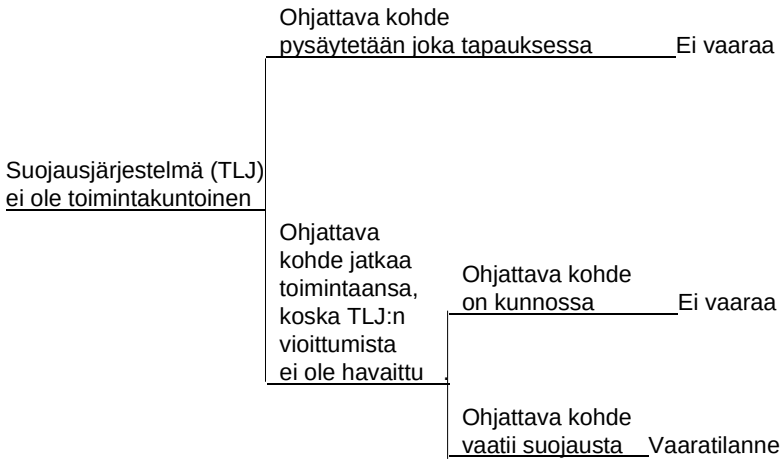
Edut	Haitat
Tunnistaa järjestelmällisesti mahdolliset vaarat ja niiden vaikutukset kaikissa projektin elinkaaren vaiheissa.	Aikaavievä. Vaatii eri aloilta asiantuntijoita, joiden on kokoonnuttava säännöllisesti.
Eliminoi virheet ja vähentää vaikutuksia.	Vaatii eri aloilta asiantuntijoita, joiden on kokoonnuttava säännöllisesti.
Auttaa suunnittelijoita ratkaistaessa varmistusten lisäämistarvetta.	Projektin kuluessa voi tulla aikataulullisia ongelmia sijoittaa tarkastelut projektiin.
Kohdistaa tarkastelua vaaralle alttiille alueille.	Vaikea päätellä, miten täydellisesti tarkastelut on saatettu loppuun.
Työkalu voi perustua tekstuaaliseen esitystapaan tai taulukkolaskennan ohjelmaan.	

5.3.4 Tapahtumapuuanalyysi

Tapahtumapuuanalyysia (Event Tree Analysis, ETA) käytettiin ensimmäisen kerran 1970-luvun alussa erään ydinvoimalaitoksen turvallisuustarkasteluissa, kun oli todettu, että vikapuu tietylle huipputapahtumalle kasvoi aivan liian monimutkaiseksi. Tapahtumapuu omittiin liike-elämässä yleisesti käytetystä päätöspuusta, jolla laaja ongelma rajataan pieniin osiin.

ETA:lla tutkitaan systemaattisesti tietystä alkutapahtumasta (inhimillinen virhe, laitteen toimintahäiriö jne.) lähtevät tapahtumaketjut ja kuvataan ne loogisen kaavion avulla. Tekniikka edellyttää riittävästi tietoa tarkasteltavasta kohteesta, muuten päätelmät tapahtumien vaikutuksista ovat hankalasti selvitettäviä.

Tapahtumapuun analysointi aloitetaan varsinaisesta vioittumisen syystä eli alkutapahtumasta ja sitä jatketaan tunnistamalla siitä aiheutuvia seurauksia eli osatapahtumia. Puu haarautuu kunkin osatapahtuman kohdalla kahteen vaihtoehtoiseen haaraan, yleensä positiiviseen ja negatiiviseen seuraukseen (Kuva 7). Tapahtumapuu konstruoidaan siis käänteisessä järjestyksessä vikapuuhun verrattuna. Yleisessä tapauksessa haarautuminen voi tapahtua useampaankin vaihtoehtoiseen reittiin.



Kuva 7. Esimerkki järjestelmätason tapahtumapuuanalyysista. Turvallisuuteen liittyvä järjestelmä (TLJ) pysyy saavuttamassaan tietyssä turvallisuuden eheystasossa vain, kun ohjattavalle kohteelle ja sen ympäristölle, mm. ihmisille ei aiheudu vaaraa sen toimimattomuuden takia.

Periaatteessa tekniikka soveltuu ohjelmiston elinkaarivaiheisiin määrittelystä integrointitesteihin, mutta ohjelmistoille suoritetuista analyyseista ei ole referoituja tuloksia. Tiedetään [Ippolito & Wallace 1995] kuitenkin ETA:n soveltuvan hyvin ajallisesti etenevien tapahtumaketjujen mallintamiseen tapauksissa, joissa tapahtumat ovat joko toisistaan riippumattomia tai syy-seuraussuhde on yksisuuntainen.

Tapahtumapuun laadinnan yksi pääongelmista on alkutapahtuman oikea valinta. Jos lopputapahtuma on tiedossa, voidaan esimerkiksi FMEA-tyylisillä tarkasteluilla kerätä vioittumistavat, jotka johtavat kyseiseen lopputapahtumaan. Yleensä kuitenkin valintaan tarvitaan kokemusperäistä tietoa aikaisempien tutkimusten, raportoitujen tapahtumien tai puhtaasti päättelyn pohjalta.

Taulukko 11. Tapahtumapuuanalyysin edut ja hyödyt ohjelmiston luotettavuusarvioinnin kannalta.

Edut	Haitat
Tapahtumapuita on helppo rakentaa. Puut ovat ymmärrettäviä. Seurausten todennäköisyydet on helppo laskea. Tietokoneavusteisia työkaluja on runsaasti saatavilla. Yhdistettävissä moneen muuhun tekniikkaan.	Monimutkaisen systeemin tapahtumapuut kasvavat suuriksi. Tapahtumien seurauksia ei ole aina helppo tunnistaa, etenkään silloin, kun ollaan syvällisellä suunnittelun tai toteutuksen tasolla.

5.3.5 Ohjelmiston oikopolkuanalyysi

Ohjelmiston oikopolkuanalyysi (engl. Software Sneak Analysis, SSA) perustuu Boeing Aerospace Companyn elektroniikalle kehittämään oikopolkuanalyysiin [Ippolito & Wallace 1995, Rankin 1973]. Tavoitteena on paikantaa odottamaton, piilevä polku tai logiikkavuo, joka tiettyjen ehtojen vallitessa tulee suoritetuksi ja joka voi johtaa vaaratilanteeseen tai estää tietyn halutun toiminnon tapahtumisen. Polku voi muodostua laitteistosta tai ohjelmistosta, operaattorin toimenpiteistä tai näiden yhdistelmistä. Oikopolut eivät ole laitteiston vikaantumisen tuloksia vaan järjestelmään tai koodiin tarkoituksettomasti sijoitettuja piileviä ehtoja, jotka voivat aiheuttaa virhetoimintoja ehtojen tullessa voimaan. Oikopolkukäsite muistuttaa nk. salaovia (engl. trap door), jotka ovat tahallisesti ohjelmaan jätettyjä koodinosia, jotka tietyillä ohjelman sisäisillä tai ulkoisilla herätteillä käynnistetään.

Oikopiirit voidaan luokitella seuraavasti [Ippolito & Wallace 1995]:

- Oikopolut virran, energian tai loogisen sekvenssin kulkeutumiseen odottamattomaan haaraa tai suuntaa pitkin.
- Oikoajoitukset, joissa tapahtumat ilmenevät odottamattomissa tai ristiriitaisissa sekvensseissä, oikoilmaisut antavat virheellisen näytön järjestelmän toimintaehdoista ja johtavat siten operaattoria harhaan.
- Oikoilmaisut antavat virheellisen kuvan järjestelmän toimintaehdoista ja johtavat siten operaattoria harhaan.

- Oiko-otsakkeet virheellisesti tai epätarkasti otsikoivat järjestelmän toimintoja, esim. syötteitä, väyliä, ohjaussekvenssejä, jne. ja johtavat siten operaattoria harhaan.

Yleisen riskianalyysin kulku (Taulukko 6) sopii myös tälle tekniikalle.

Peyton & Hess [1985] kuvaavat tekniikan kulkua seuraavasti:

1. Kerätään kaikki järjestelmään kuuluva informaatio.
2. Rakennetaan tietokanta kaikille paikoille, joissa muuttujia tai asianmukaista tietoa käytetään, sekä kaikille alirutiinikutsuille ja niille nimetyille synonyymeille.
3. Esitetään ohjelman logiikka sähköpiirisymbooleilla kuvatuilla verkkopuilla.
4. Tarkastellaan ohjelman dataa verkkopuun avulla. Tavoitteena on tunnistaa kaikki sellaiset kohdat, jotka voivat johtaa ongelmien lähteille. Niitä ovat mm. suunnitteludokumentit, joita tarkemmin tarkastellessa voidaan jäljittää ongelmia, tai osoittimet, jotka voivat johtaa tavallisesta poikkeavien ohjausvoiden tunnistamisiin.

Kuten kohdasta neljä saatetaan havaita, menetelmä ei ole kovin täsmällinen. Tarkistuslistoja käyttämällä voidaan intuitiviisesti mahdollisesti päätellä ongelmakohtien olemassaolosta. Peyton & Hess [1985] väittävätkin, että tekniikka tunnistaa tehokkaasti sellaisiakin virheitä, joita ei ole löydetty manuaalisissa tarkastuksissa tai standardeilla todennusmenetelmillä. Käyttö edellyttää kuitenkin viimeisteltyä ohjelmistodokumentaatiota sekä kyvykästä ja kokenutta tarkastelijaa.

Tekniikka soveltuu paremmin toimintavarmuuden analysointiin kuin turvallisuuden, sillä on epätodennäköistä, että oikopolkuanalyysilla voidaan havaita vakavia virheitä [Leveson 1986]. Edelleen Leveson väittää, että tekniikkaa käytetään harvoin eikä se ole kovin tehokas.

Lähteen Ippolito & Wallace [1995] kirjallisuustarkastelun mukaan ohjelmiston oikopolkuanalyysi soveltuu periaatteessa myös kaikkiin järjestelmän tai ohjelmiston kehityksen elinkaarivaiheisiin vaatimusmäärittelystä integrointivaiheisiin em. toteutusvaiheen lisäksi.

Pulkkinen kuvaa [1993] analyysin kulkua kooditarkastelussa. Aluksi haetaan (ehkä tarvittaessa jollakin toisella vaara-analyysin tekniikalla) ne ohjelman ulostulot, jotka voivat johtaa vaaratilanteeseen tai epätoivottuun ohjaukseen. Näiden ulostulojen syyt haetaan etenemällä taaksepäin ohjelmarakenteessa ottaen huomioon normaalit ohjelman toi-

mintaan liittyvät loogiset ehdot siihen asti, kun ohjelman sisäänmenot on saavutettu. Tekniikkaan voidaan ottaa huomioon laitevioista aiheutuneet lisähäiriöt tai datan eheyden menettämisen aiheuttamat tapahtumat. Ainakin kooditasolla tekniikka muistuttaa ohjelmiston vikapuuanalyysia.

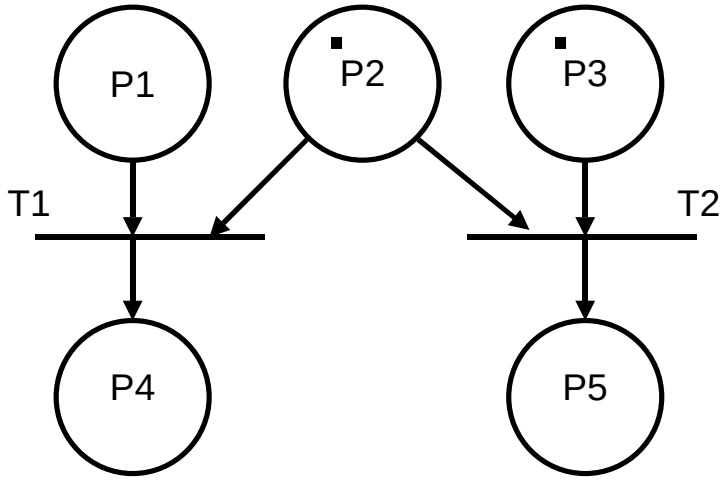
Ohjelmiston oikopolkuanalyysi on luotettavuuden tarkastelutekniikka siinä mielessä, että se voidaan yhdistää vikapuuanalyysiin. Vikapuuanalyysia kannattaa täydentää oikopolkujen tunnistamisella silloin, kun on erityistä syytä epäillä ylimääräisen tai piilossa olevan ohjelmakoodin ’unohtuneen’ ohjelmaan. Lähde [Ippolito & Wallace 1995] suosittaa kolmea kriteeriä tällaisten piilokoodien mahdolliselle esiintymiselle:

1. Järjestelmä on monimutkainen – mitä kompleksisempi järjestelmä on, sitä useampia piilossa olevia oikopolkuja se sisältää.
2. Organisaatio on laaja ja monimutkainen – on vaikeaa täsmällisesti määrittää ja varmistaa liittymien ristiriidattomuutta, jos organisaatio esimerkiksi sisältää useita alihankkijoita.
3. Nopeasti muuttuva teknologia – uusien järjestelmien analyyseja ja testauksia joudutaan usein vähentämään.

5.3.6 Petriverkot

Carl Petri kehitti 1961 matemaattista järjestelmämallinnusta varten menetelmän, missä järjestelmä mallinnetaan tiloilla ja niiden välisillä tapahtumilla. Matemaattisen esitystavan lisäksi Petriverkot voidaan esittää graafisesti (Kuva 8), jolloin tiloja (paikkoja) kuvataan ympyröillä, tapahtumia (transitioita) palkeilla, syötteitä (paikoista palkeille) ja tulosteita (palkeita paikoille) nuolilla sekä merkeillä, jotka ovat Petrikaavioiden yhteydessä käytettyjä alkiomerkkejä (pisteet – osoitus ”tosi”-tilanteesta).

Annetuilla alkuehdoilla Petriverkko voidaan suorittaa ja tarkastella siitä, miten tilat ovat saavutettavissa. Vaara-analyyseissa Petriverkon tilat vastaavat potentiaalisia vaaratiloja, joiden saavuttamista tutkitaan. Erityisesti tarkastellaan ohjelman suoritusaikaisia suojaavia toimenpiteitä, joilla estetään tapahtuman kulku kohti vaaratilaa. Tekniikkaa onkin hyödynnetty sopivien suoritusaikaisten mekanismien, kuten vahtikoirien (ajastustekijöiden, aikarajojen) suunnittelussa [Storey 1996].



Kuva 8. Petrikaavio. Ympyrät P1–P5 kuvaavat paikkoja ja palkit T1–T2 transitioita. Ympyröissä P2–P3 on merkit, jotka ilmaisevat paikan olevan tosi. Koska P1:ssä ei ole merkkiä, niin huolimatta P2:n merkistä, merkki ei periydy P4:lle. Paikkojen P2 ja P3 merkit periytyvät paikkaan P5 [Leveson 1995].

Petriverkko on ensi sijassa suunnittelumenetelmä, jonka käyttäminen vaara-analyyseissa edellyttäisi ohjelmiston suunnittelemista Petriverkkomenetelmällä. Tämä vaatii asiantuntemusta ja on työlästä. Kaavio voi helposti laajentua liikaa, jos halutaan tarkastella kaikkia järjestelmän tiloja. Liiallinen laajentuminen merkitsee myös tarkastelun hankaloitumista.

Petriverkon tehokkuus tarkastella järjestelmän dynaamista käyttäytymistä on ominaisuus, jota tapahtumapuuanalyysia lukuun ottamatta ei ole edellisissä kohdissa esitetyillä analyysitekniikoilla. Siksi sen käyttämistä vaara-analyyseissa kannattaa harkita huolimatta tekniikan työläydestä ja laajentumisalttiudesta. Eräs hyödyntämisnäkökohta on tekniikan kohdentaminen suppeisiin, vain kaikkein kriittisimpiin ohjelmistomoduuleihin, joihin on todettu liittyvän mahdollisia dynaamisia tilansiirron virhetilanteita.

Ippoliton & Wallacen [1995] kirjallisuuskatsauksen mukaan Petriverkon käyttäminen dynaamisten ominaisuuksien suunnitteluun ja tarkasteluun kehityksen varhaisissa vaiheissa kannattaa nimenomaan muutostöiden aiheuttamien kustannussäästöjen takia verrattuna muutosten tekemiseen esimerkiksi testausten jälkeen. Leveson [1995] suosittaa tekniikkaa vielä yksityiskohtaiseen suunnitteluun. Edelleen hän kuvaa tekniikan käyttämistä vaara-analyysina, mikä muistuttaa yleistä riskianalyysin kulkua (Taulukko 6). Vaarat ja niiden riskitasot tunnistetaan ensin jollakin muulla vaara-analyysitekniikalla. Jos vaaratapahtumilla on dynaamisia piirteitä, laaditaan niitä vastaavat Petrikaaviot ja

tarkastellaan vaarallisten tilojen saavutettavuutta. Sopivia tunnistamismenetelmiä ovat ainakin vika- ja vaikutustyyliset tekniikat.

Leveson on käsitellyt Petriverkon dynaamisen tarkastelutavan yhdistämistä sekä vikapuihin [1987] että tilakoneisiin [1995]. Edellisessä tavoitteena on yhdistää ajastustapahtumat vikapuuanalyysiin, koska sillä ei pystytä riittävän tehokkaasti tarkastelemaan moninkertaisia ja useasta eri paikasta johtuvia virhetilanteita. Lähestymistapojen yhdistäminen on mahdollista, koska kummatkin ovat formaaleja kuvaustekniikoita. Vikapuun liittäminen Petriverkkoanalyysiin pienentää verkon monimutkaisuutta ja työmäärää; toisaalta kahden tekniikan käyttö johtaa todennäköisesti tarkempaan virheiden tunnistamiseen kuin vain jommalla kummalla tekniikalla tehtynä.

Petriverkon yhdistämisessä tilakoneeseen (State Machine Hazard Analysis, SMHA) on kyse ohjelmiston ja laitteiston rajapinnalla esiintyvien virheiden ja virhetoimintojen mallintamisesta ja tunnistamisesta.Periaatteessa Petriverkkotekniikka on toteutettavissa vaara-analyysina tai sen apuna, erityisesti koska SMHA-tekniikalle on automaattisiakin työkaluja. Mutta se, miten tarkoin tekniikka voidaan rajata käsittämään vain tiettyä vaatimus-, arkkitehtuuriosa- tai moduulijoukkoa ottamatta huomioon koko järjestelmää, eli mikä merkitys on sillä, että tietty haku jätetään esimerkiksi moduulin rajapinnalle, jää jatkotutkimuksen määritettäväksi.

5.4 Muut luotettavuusanalyysit

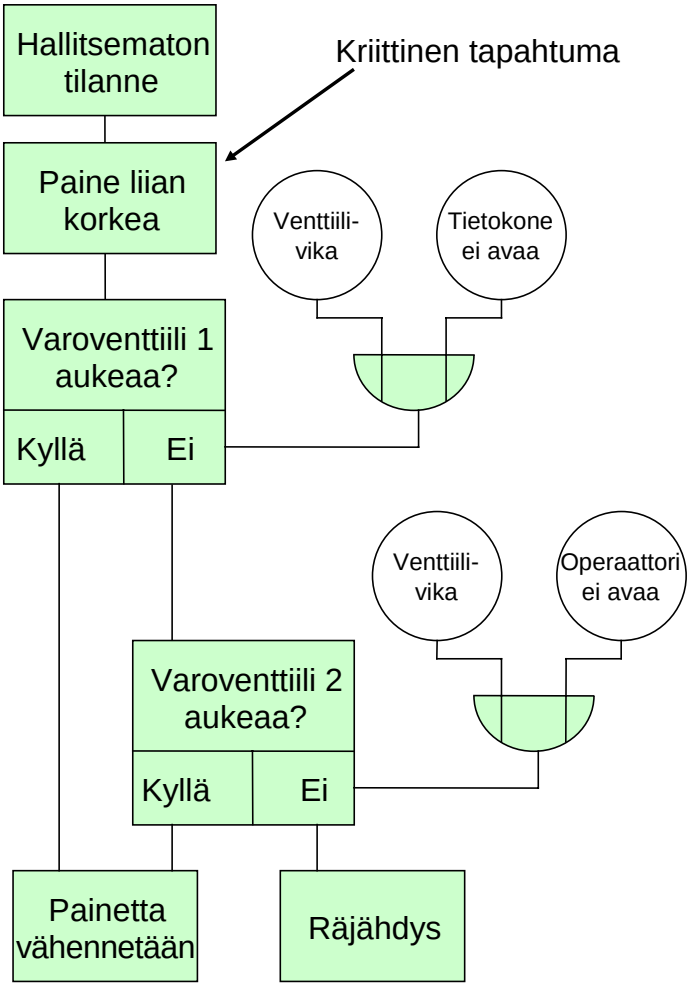
Syy-seurauskaavion tavoitteena on mallintaa tapahtumasekvenssit kaaviomuotoon. Menetelmällä saadaan informoidusti kuvattua monimutkaiset yhdistelmät syysuhteesta tapahtumavirrassa. Se on eräänlainen vikapuun, tapahtumapuun ja tilamallin yhdistelmä. Menetelmässä jäljitetään kriittisestä tapahtumasta mahdolliset tapahtumaketjut sekä induktiivisesti että deduktiivisesti (Kuva 9). Kaaviossa otetaan huomioon sekä aikariippuvuudet että kausaaliset tekijät. Deduktiivinen jäljittäminen vastaa vikapuuanalyysia, induktiivinen muistuttaa läheisesti tapahtumapuuanalyysia, kuten Kuvan 9 esimerkkikin osoittaa.

Syy-seurauskaavio on malleista monipuolisin ja soveltuu monimutkaistenkin semanttisten syysuhteiden kuvaamiseen. Analyysi on kvantifioitavissa, ja sille löytyy työkaluja. Kaaviosta tulee helposti hyvin suuri [Bishop 1990].

Korjaavan ja ennakoivan ylläpidon etukäteissuunnitteluun liittyy h*uolto- ja vaikutusanalyysi* [Lyytikäinen 1987]. Sen avulla voidaan järjestelmällisesti tunnistaa ylläpitoon vaikuttavia tekijöitä huollettavuuden ja huoltovarmuuden parantamiseksi, ja sillä voidaan kehittää ylläpitoa ja optimoida varaosien tarvetta. Tekniikka liittyy läheisesti laitteistoihin; ohjelmistoihin sitä ei tiedetä käytetyn, mutta se on tutkimuksen ja kehityksen

kannalta varteenotettava menetelmä. Ylläpidettävyys on yksi ohjelmiston elinkaaren tärkeimmistä ja samalla kalleimmista vaiheista.

Suositeltavinta on käyttää tekniikkaa varhaisessa vaiheessa, mahdollisesti FMECA:n yhteydessä, jolloin järjestelmän käyttövarmuuden kannalta kriittiset komponentit käsitellään.



Kuva 9. Syy-seurauskaavion laatiminen aloitetaan kriittisestä tapahtumasta. Symboleita ovat seurausten mallintamisessa mm. Kyllä-Ei-kaaviot ja syiden mallintamisessa vikapuun kaaviot, mm. TAI-portteineen, kuten kuvassa.

Elinjakson kustannusanalyysi on normaalia kustannuslaskentaa, johon otetaan huomioon käyttövarmuus- ja ylläpitotekijät koko laitteiston käyttöiän ajalta [Lyytikäinen 1987]. Tekniikan soveltaminen ohjelmistolle on mahdollista.

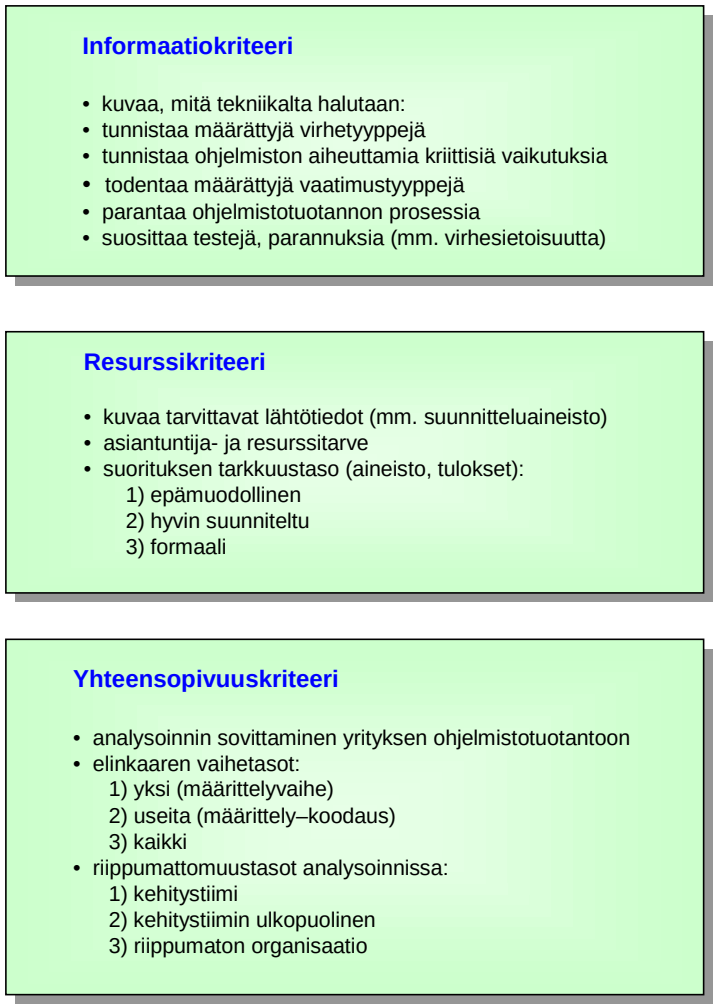
Potentiaalisten ongelmien analyysin tavoitteena on löytää kohteen keskeisimmät ongelma-alueet sekä keskeisimpiin vaaroihin liittyvät onnettomuustekijät. Periaatteena on, että ideainhakumenetelmillä etsitään kohteen onnettomuusvaaroja ja luokitellaan ne esiintymistaajuuden ja seurausten vakavuuden mukaan. Tämän jälkeen analysoidaan keskeisempien vaarojen syyt ja seuraukset.

Leveson [1995] kuvaa muutamia ihmisen tekemiin tehtäviin liittyviä luotettavuuden analyysitekniikoita: toimenpideanalyysia, operaattorin tehtäväanalyaysi, toimintovirheanalyysia ja työturvallisuusanalyysia. Hän myös antaa useita viitteitä tekniikkoja kuvaaviin lähteisiin. Kolme ensimmäistä tekniikkaa muistuttavat toteutus- ja esitystavaltaan FMEA:ta, viimeksi mainittu HAZOP:tä, missä hakuprosessi etenee peräkkäisten työtehtävien mukana.

6. Analyysikriteerit

Arviointimenetelmät- ja tekniikat eivät ole täydellisiä edes toisiinsa yhdistettyinä. Virheitä tunnistavat tekniikat on mahdollista kohdistaa vain joihinkin virheisiin, koska aika-, henkilö- tai välineresurssit ovat riittämättömiä jokaisessa elinkaaren vaiheessa käytettäviksi. Vaikka arviointimenetelmien käyttäminen ei takaa kaikkien järjestelmässä vallitsevien virheiden ja puutteiden tunnistamista, se kasvattaa luottamusta systeemin virheettömään toimintaan.

Tässä luvussa määritetään kriteereitä, jotka tukevat ohjelmiston luotettavuuden analysoimista ja arviointitekniikoiden valitsemista. Kohdassa 5.3 sivuttiin aihepiiriä mm. selvityksillä tekniikoiden eduista ja haitoista sekä soveltumisesta elinkaaren vaiheisiin ja työtapoihin.



Kuva 10. Luotettavuuden analyysitekniikoiden valintakriteerit.

Tekniikoista halutaan tietää niiden tulosten hyödyllisyydestä, tekemisen kyvykkyydestä ja vastaavuudesta erilaisiin ohjeisiin ja standardeihin. Näitä toiveet täyttäviä luotettavuuden analyysitekniikkojen valintakriteeriteitä on useita, ryhmittelytapojakin voidaan kehittää erilaisia.

Kun halutaan tietää tekniikan tuottama informaatio, kyse on informaatiokriteeristä (Kuva 10). Kun halutaan tietää, millä tavoin ja kuinka tehokkaasti tekniikan tuottama informaatio saadaan hankituksi, kyse on resurssikriteeristä. Edelleen, haluttaessa tietää, miten tekniikka sopii ympäristöönsä, esimerkiksi yrityskulttuurin ohjelmistotuotantoon todennuksineen ja kelpoistuksineen, sekä muihin käytettyihin luotettavuuden analysointitekniikkoihin, kyse on yhteensopivuuskriteeristä.

6.1 Informaatiokriteeri

Informaatiokriteeri kuvaa niitä tuloksia, joita luotettavuuden analysoinnilta halutaan sekä yksittäisen analyysitekniikan että jonkun kokonaisuuden, kuten integroidun menetelmän, kannalta. Informaatiota on tietysti ensi sijassa luotettavuusarvio ohjelmiston virheettömyydestä, mikä johtaa luotettavuusattribuuttien tavoiteasetteluun sekä viittaa informaatioon mm. tekniikoilla tunnistettavista virhetyypeistä.

Taulukko 12 esittää vertailun staattisilla analysaattoreilla tunnistettavista virhetyypeistä, mikä tunnistamistapa ei kuitenkaan ole keskeisintä luotettavuuden analyysitekniikoilla vaan mahdollisten virhetoimintojen tunnistaminen. Taulukon esittämät virhetyypit ovat ensi sijassa niiden detaljitason syitä. Esimerkiksi ajastusvirheet ovat vaikeita tunnistettavia ohjelmiston FMECA:lla, koska sen suoritustapa ei ole dynaaminen. Parametri- ja raja-arvovirheet voidaan tarkastaa luotettavuustekniikoilla normaalin tarkastuksen tapaan. Syntaksi- ja datavirheiden tarkastelutapa muistuttaa luotettavuuden analyysitekniikoilla myös tavallisia tarkastustekniikoita. Tunnistettavuutta esittää Taulukko 13.

Tekniikoiden avulla 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. (ks. yhteensopivuuskriteeri).

Taulukko 12. Luotettavuuden analyysitekniikoiden soveltuvuus virheiden tunnistamiseen fyysisessä (vrt. toiminnallisessa) tarkastelutavassa. Ei: tekniikka ei sovellu, S: tekniikkaa suositellaan varauksin, HS: tekniikka on hyvin suositeltava.

Tekniikat	Virhetyypit				
	Ajastus	Logiikka	Parametri	Syntaksi	Data
Vika- ja vaikutus	Ei	S	S	S	S
Vikapuu	Ei	HS	S	S	S
Poikkeamatarkastelu	Ei	S	Ei	Ei	Ei
Tapahtumapuu	Ei	Ei	Ei	Ei	Ei
Oikopolku	Ei	Ei	Ei	Ei	Ei
Petriverkko	HS	HS	S	S	S

Taulukko 13. Luotettavuuden analyysitekniikoiden soveltuvuus kvalitatiiviseen ja kvantitatiiviseen arviointiin sekä syiden, seurausten, havainto- ja estomekanismien tunnistamiseen. Ei: tekniikka ei sovellu, S: tekniikkaa suositellaan varauksin, HS: tekniikka on hyvin suositeltava.

Tekniikat	Kvalitat.	Kvantitat.	Vioittumistapojen tunnistettavuus			
			Syyt	Seuraukset	Havainto-mekan.	Esto-mekan.
Vika- ja vaikutus	HS	Ei	S	HS	HS	HS
Vikapuu	HS	HS	HS	Ei	S	S
Poikkeamatarkastelu	HS	Ei	S	HS	HS	HS
Tapahtumapuu	Ei	HS	Ei	HS	S	S
Oikopolku	S	EI	S	S	S	S
Petriverkko	S	Ei	S	Ei	S	S

6.2 Resurssikriteeri

Resurssikriteeri kuvaa ne ominaisuudet, joita luotettavuuden analyysitekniikoilta vaaditaan tarvittavan informaation tuottamiseksi kyseisellä tekniikalla. Kriteeriin sisältyvät tekniikan vaatimat ohjelmistotuotannon dokumentit sekä asiantuntija- ja muun henkilöstön tarve. Tarvittava asiantuntemus riippuu tekniikan käytön helppoudesta, työkalujen saannista ja muista vastaavista ominaisuuksista (Taulukko 14). Arviointi helppokäyttöisyydestä ei ole kovin yksinkertaista. Ohjelmiston vikapuuanalysointi vaatii

kvantitatiivisesti perehtymistä ja kouluttautumista aiheeseen. Sama pätee Levesonin [1983, 1991] esittämään kvalitatiiviseen tekniikkaan. Myös mahdollisten virheiden tunnistaminen vaatii harjaannusta, mutta ohjelmiston FTA:n periaatteellinen kvalitatiivinen suoritustapa on yksinkertainen. 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.

Taulukko 14. Ohjelmiston luotettavuuden analyysitekniikoiden ominaisuuksia. Kuvattujen ominaisuuksien arviointi on hyvin subjektiivista ja projektikohtaista. Oikeilla tekniikan työmenetelmien valinnoilla päästään yksinkertaisiin ja kuitenkin onnistuneisiin ratkaisuihin. Ei: tekniikka ei sovellu, V: voi olla osittain, ON: on hyvin soveltuva.

Tekniikat	Ominaisuudet				
	Helppo-käyttöinen	Syste-maattinen	Työkalu-saanti	Informoi-tavuus	Vähä-töinen
Vika- ja vaikutus	V	ON	ON	V	Ei
Vikapuu	V	V	ON	ON	Ei
Poikkeamatarkastelu	V	ON	ON	V	Ei
Tapahtumapuu	V	V	ON	ON	Ei
Oikopolku	–	–	Ei	–	Ei
Petriverkko	V	V	ON	V	Ei

Tekniikat voidaan myös suorittaa eri tarkkuustasoilla riippuen tarkasteltavan kohteen laajuudesta, käytettävissä olevista resursseista sekä kohteen riskitasosta. Suorituksen tarkkuus jaetaan kolmeen perustasoon seuraavasti:

1. epäformaali analysointitapa
2. hyvin suunniteltu analysointitapa
3. formaali analysointitapa.

Ensimmäisessä ja yksinkertaisimmassa suoritustasossa analysoinnin suunnittelu on epäformaalia ilman tarkkaa suunnitteluprosessia ja -dokumentaatiota. Myös analysointi suoritetaan sekä tulokset esitetään epämuodollisesti dokumentoimatta niitä muodollisen tarkasti. Epäformaali analysointitapa sopii ohjelmistotuotannon projektihenkilöstön itsensä suorittamiin tarkasteluihin, jolloin mm. määrittely- ja suunnitteluaineistoa ollaan vasta tekemässä. Tässä tapauksessa tulokset jäävät ensi sijassa kehitysprosessin käyttöön. Myös alimmilla luotettavuuden vaatimustasoilla yksinkertaisin suoritustaso on riittävä myös riippumattomalle analysointiryhmälle.

Toisessa tasossa analysointi suoritetaan huolellisesti tehdyn suunnitelman ja metodiikan pohjalta. Tulokset perustellaan ja dokumentoidaan tarkasti. Taso on normaali riippumattomalle analysointiryhmälle, joiden tehtävänä on yksiselitteisesti ja ymmärrettävästi perustella tulokset.

Kolmannessa eli tarkimmassa tasossa analysoinnissa noudatetaan formaalisti määriteltyä suunnitelmaa. Tämä edellyttää sekä ohjelmistotuotannon prosessivaiheiden formaalisointia että sopivan työkalun käyttämistä. Tulokset kuvataan yksityiskohtaisesti ja perustellaan kuvaamalla esimerkkejä samankaltaisista järjestelmistä.

6.3 Yhteensopivuuskriteeri

Yhteensopivuuskriteeri kuvaa ne luotettavuuden analysointitekniikoiden ominaisuudet, jotka tarvitaan tekniikan sovittamiseksi yrityksen ohjelmistotuotantoon. Nykykäytäntöä kuvaavat yritysten laatudokumentit ja erilaiset tuotantoon kohdistuvat säännöt ja ohjeet, mm. ohjelmointisäännöt sekä tuotosdokumentit eli aineisto, jota analyyseissa tarvitaan. Niitä on tarkasteltu etenkin luvussa 4.

Taulukko 15. Luotettavuuden analyysitekniikoiden soveltuvuus ohjelmistotuotannon elinkaaren vaiheisiin. Ei: tekniikka ei sovellu, S: tekniikkaa suositellaan varauksin, HS: hyvin suositeltava.

Tekniikat	Elinkaarivaiheet				
	Määrittely	Suunnit-telu	Koodaus	Testaus-vaiheet	Käyttö & muutokset
Vika- ja vaikutus	HS	S	S	Ei	S
Vikapuu	HS	S	S	S	S
Poikkeamatarkastelu	S	S	Ei	Ei	S
Tapahtumapuu	S	HS	Ei	Ei	S
Oikopolku	Ei	HS	S	Ei	Ei
Petriverkko	HS	HS	S	Ei	Ei

Yhteensopivuuskriteeri yrityksen ohjelmistotuotantoon riippuu käytetyistä työmetodeista elinkaaren eri vaiheissa. Taulukko 15 kuvaa tekniikoiden soveltuvuutta ohjelmistotuotannon elinkaareen. Taulukon arvoissa on pyritty ilmaisemaan soveltuvuus riippumattomasti muista tekniikoista ja aikaisemmin tai rinnan tehdyistä analyyseista – ikään kuin tekniikka suoritettaisiin ainoana menetelmänä.

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.

Seuraavassa esitetään yleinen kolmitasoinen luokittelu analyysien sopivuudesta elinkaaren vaiheisiin:

1. yksi varhainen elinkaarivaihe
2. kaikki vaiheet alusta suunniteltuvaiheisiin
3. kaikki vaiheet alusta toteutusvaiheeseen.

Yksinkertaisimmillaan analysoinnit tehdään yhdessä vaiheessa ennen moduulisuunnittelun alkamista. Mahdollisia vaiheita ovat määrittely ja arkkitehtuurisuunnittelu, jotka myös usein on yhdistetty yhdeksi elinkaarivaiheeksi. Toisessa tasossa analysoidaan kaikissa vaiheissa määrittelystä moduulisuunnitteluun (se mukaan lukien). Kolmas taso sisältää kaikki yrityksen ohjelmistotuotannon elinkaaren vaiheet. Analysoinnit ovat normaaleita tarkasteluita kaikissa vaiheissa määrittelystä toteutukseen ja lisäksi tuloksia päivitetään käytön aikana lähinnä muutosten yhteydessä.

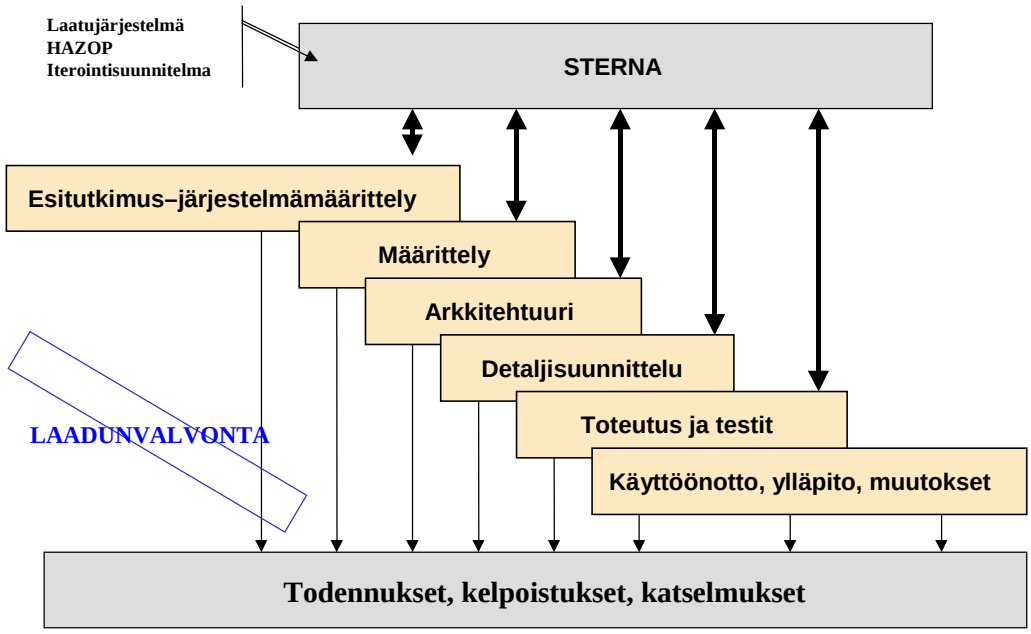
Riippumattomuus analysoinnin tekijöiden ja kehitystyön tekijöiden välillä jaetaan seuraaviin tasoihin:

1. ei riippumattomuutta
2. riippumaton ryhmä
3. riippumaton organisaatio.

Yksinkertaisimmillaan analysoinnit tekevät kehitystyöhön osallistuvat ryhmän jäsenet. Seuraavaksi korkein taso on erillinen kehitystyöstä riippumaton ryhmä ja korkein taso riippumaton organisaatio. Ylimmässä tasossa analysoinnin tekijöillä on merkittävä kokemus vastaavista arvioinnin kohteena olevista systeemeistä siten, että he kykenevät hyvin itsenäiseen suoritukseen.

7. Ohjelmiston luotettavuuden kelpoistusmenetelmä

Tässä luvussa esitetään ohjelmiston luotettavuuden tarkastelumenetelmä, jolle annetaan työnimeksi Stérna (tiira). Menetelmä perustuu ensi sijassa ohjelmistokehityksen varhaisempien vaiheiden tarkasteluun sekä näiden tarkasteluiden ohjaavaan ja luotettavuutta parantavaan vaikutukseen (Kuva 11).



Kuva 11. Stérna-malli on kiinteä osa ohjelmistotuotannon elinkaarta. Analyysien tuloksena saadaan täydennettyä ja korjattua ohjelmistotuotannon dokumentointia. Täydennysten ja viimeistelyn jälkeen suoritetaan laadunvarmistustoimenpiteitä.

Vaatimus kustannustehokkuudesta yltää kaikkiin luotettavuus- ja riskitasoihin alemmista käytettävyyden tai ylläpidon tasoista korkeimpiin turvallisuuden eheystasoihin. Hyvään kustannustehokkuuteen ei päästä liittämällä osatekijät sattumanvaraisesti yhteen vaan harkitsemalla tiettyä proseduuria, noudattamalla erilaisia vaihtoehtoja sekä ottamalla huomioon tarkastelukohde ja yritysympäristö mahdollisine viiteryhmineen.

Tässä luvussa tarkastellaan tietyn kaavan mukaisesti ohjautuvaa ohjelmiston luotettavuuden analyysimenetelmää, joka välittää analysoijalle suhtautumista vaiheittaiseen tarkasteluun ja asennoitumista tarkastelun sisältöön. Luotettavuuden analysoinneissa on

aina kyse ensi sijassa ajattelutavasta, joka oikeanlaisen suhtautumisen ja asennoitumisen kautta siirtyy ohjelmiston kehittäjille ja ylläpitäjille niin, että rakennetaan virheetön ohjelmisto jo määrittelyn ja suunnittelun aikana eikä ainoastaan testausten, katselmusten ja tarkastusten myötävaikutuksesta.

7.1 Menetelmän periaatteet

Stérnan tavoitteena on tunnistaa ohjelmistoista aiheutuvia vaaroja ja riskejä, niiden merkittävyyttä ja niihin johtavia syitä. Arviointi kohdistuu luotettavuusattribuuttien merkittävyyteen vaatimusten määrittelyn ja toteuttamisen kannalta sekä painottaa analyyseja, testejä ja katselmuksia kriittisiin kohteisiin.

Tässä käytetty ohjelmistokehityksen elinkaarimalli vastaa hieman modifioituna yleisiä viitemalleja useimmissa alan standardeissa ja ohjeissa [mm. IEC 61508 1999]. Varsinainen ohjelmiston luotettavuuden analysoinnin aloitusvaihe on ohjelmiston toiminnallinen määrittely (Kuva 11), mitä ennen on jo tehty analyysin vaatimat valmistelutyöt. Valmistelua vaativat mm. tarkastelukohteen määrittäminen, tiettyjen yritys- ja/tai projektikohtaisten taulukoiden ja tarkistuslistojen kokoaminen tai valmistelu sekä analyysille kohdentuvien tavoitteiden asettaminen.

Tässä Stérna kulkee vesiputousmallia myötäillen mutta ei riipu tietystä elinkaarimallin vaiheistuksesta. Kyse on dokumentaatiomallista, joka kokoaa tietyt ohjelmiston luotettavuudelle tärkeät ominaisuudet yhteen ja jota ilman analysointia ei ole mahdollista. Usein ohjelmistopohjaisissa järjestelmissä ei tehdä erikseen ohjelmiston vaatimusmäärittelyn dokumenttia tai ei ole erikseen vaihetta ohjelmiston arkkitehtuurivaiheelle. Näissä tapauksissa varsinaiset ohjelmistotehtävät, määrittelyt ja arkkitehtuurit tehdään osana järjestelmän elinkaarivaiheita. Ohjelmisto määritellään järjestelmän arkkitehtuurisuunnittelussa ja ohjelmiston arkkitehtuuri osana osajärjestelmän suunnittelua.

Vastaavasti luotettavuustarkastelua ei välttämättä tarvitse erottaa laitteiston luotettavuustarkastelusta. Integroitaessa nämä kaksi analyysiä samaan tarkasteluun lähtökohtana pitää olla tietysti funktionaalinen tarkastelu laitteistotarkastelun sijaan. Myöskään suoraviivainen vesiputousmalli ei ole edellytys Stérnalle. Malli saa olla inkrementaalinen, iteroiva tai spiraalinen.

Jos tiettyä ohjelmistoprojektia ei aikaisemmin ole analysoitu eli tarkastelut eivät voi alkaa tietyistä etukäteen kohdennetuista tapahtumista, esimerkiksi poikkeamatarkastelun tuloksista, valmisteluvaiheessa tulisi systemaattisesti tarkastella kaikki ohjelmiston toiminnallista määrittelyvaihetta ylemmät tasot, joita tässä kutsutaan yhteisnimikkeellä *järjestelmätasot*.

Stérna-mallissa nämä järjestelmätasoihin liittyvät valmistelutyöt tehdään esitutkimus-järjestelmäanalyysin vaiheessa. Sille malli ei suosittele erityistä tarkastelutekniikkaa, mutta varsinainen aloitusvaihe eli ohjelmiston toiminnallisten vaatimusten märittelyvaiheen tarkastelu sopii parhaiten vika- ja vaikutustyyliselle perustekniikalle. Jos kriittiset kohteet ovat hyvin tiedossa eikä niihin haluta enää erityistä panostusta, Stérna kohdistuu yksinomaan syiden tunnistamiseen ja suositusten määrittämiseen.

Stérna määrittelee seuraavan vaiheen eli ohjelmiston arkkitehtuurisuunnittelun alkutapahtumat. Niistä yleisemmin käytetään edellisessä vaiheessa tunnistettuja merkittävimpiin vioittumistapoihin johtavia syitä tai myös havainto- tai estomekanismien luotettavuuteen ja häiriöihin liittyviä tapahtumia. Tapahtumavirtaa seurataan vastavirtaan eli virheiden lähteille läpi koko ohjelmiston suunnittelun ja toteutuksen. Lisäksi myöhempien vaiheiden luotettavuustarkasteluilla tulisi olla ominaisuuksia, joilla voidaan selvittää mm. määrittelyn ristiriitaisuudet.

Arkkitehtuurivaiheen merkittävin tarkastelukohde ovat mahdollisten rinnakkaisten osuuksien yhteisvirheet ja riskiä vähentävien toimintojen liittäminen suunnitteluun. Yhteisvikailmiötä muistuttaa läheisesti usean vaatimuksen kohdistuminen tiettyihin ohjelmistokomponentteihin, jolloin näiden merkittävyys kasvaa. Erityisesti näissä tapauksissa on tarkasteltava komponenttien riittävän yksinkertaista toteuttamista. Arkkitehtuurivaiheessa viimeistään käsitellään myös yleensä uudelleenkäytettäviä ohjelmistoja ja valmisohjelmistoja (esimerkiksi COTS), joista ei ole dokumentointia mm. riittävän luotettavuuden määrittämiseksi.

Stérna voidaan suorittaa ryhmätyönä jonkin toimintamallin mukaan, tai yksittäinen tarkastelija analysoi tietyllä yritykselle tai ohjelmistoprojektille sopivalla riippumattomuusasteella (ks. kohta 6.3). Mallissa ei oteta kantaa ryhmätyöskentelylle sopiviin toimintatapoihin tai henkilörooleihin. Näihin asioihin voi tutustua esimerkiksi viitteessä [Def Stan 00-58 1996], jossa esitetään malli poikkeamatarkastelun ryhmätyöskentelylle.

7.2 Avainlauseet

Stérna tarkastelee vaatimusten luotettavuusominaisuuksia ensi sijassa mahdollisten ei-toivottujen seurausten kannalta. Tällä tavoin selvitetään toimintojen kriittisyydet siten, että vaatimusten määrittely ja suunnitteluponnistukset voidaan tehokkaammin kohdistaa.

Taulukko 16. Luotettavuusattribuuttien avainlauseet vaatimusten määrittelyvaiheessa ja arkkitehtuurisuunnittelun vaiheessa.

Laatu	Avainlauseet
Toiminnallisuus (vrt. ohjelmiston virhetoiminto)	Toimintoa ei suoriteta määritellysti kaikissa toimintatiloissa. Toiminto alustetaan puutteellisesti tai virheellisesti ennen suoritusta. Toiminto suoritetaan ennen kuin käynnistysehdot ovat täyttyneet. Toiminto ei käynnisty. Toiminnon suorittaminen jatkuu, vaikka päättymisehdot ovat täyttyneet. Toiminto päättyy, vaikka päättymisehdot eivät ole täyttyneet. Toiminto suoritetaan väärässä toimintatilassa.
Toimintavarmuus	Ohjelmisto ei ole riittävän luotettava tai on tarpeettoman luotettava. Ohjelmisto ei vikaannu hallitusti vaan vikaantuu myös kriittisiltä osilta. Ohjelmiston vikasietoisuuden vaatimukset eivät täyty tai niitä ei ole Ohjelmistossa on virhetoiminta. Laitteisto vikaantuu. Ohjelmiston virhetoiminta etenee yllättäviin toimintoihin tai prosesseihin. Ohjelmisto ei onnistu palautua virhetoiminnasta. Virhetoiminnoista ei ilmoiteta tai raportoida operaattorille. Ohjelmisto ei ilmaise epäasiallista toimintatapaa. Data siirretään väärään toimintoon tai prosessiin.
Ylläpidettävyys	Uudelleenkäytettävyyden vaatimukset eivät täyty tai niitä ei ole. Ohjelmistomuutoksien vaatimukset ovat puutteelliset. Tuotehallinta on puutteellinen. Toiminto ei ole verifioitava. Toiminto ei ole mukautuva.
Turvallisuus	Ohjelmisto aiheuttaa järjestelmän joutumisen vaaralliseen tilaan. Ohjelmisto ei siirrä järjestelmää vaarallisesta vaarattomaan tilaan. Ohjelmisto ei käynnistä turvajärjestelmää. Ohjelmisto ei tunnista vaarallisia tiloja.

Järjestelmällisyydellä saadaan aikaan kattava analyysi. Sopivien ja riittävän kattavien vioittumistapojen valinta on järjestelmällisen riskianalyysin tekniikan ongelmallisin tehtävä. Valintaa tukemaan on laadittu avainsanoja [Def Stan 00-58 1996] ja -lauseita [Lawrence & Gallager 1997].

Stérnassa kriittiset vaikutukset selvitetään avainlauseiden avulla. Taulukko 16 esittää avainlauseet luotettavuusattribuuteille ja toiminnallisuudelle sekä niitä läheisille attribuuteille. Toiminnallisuus muistuttaa virhetoimintoa, robustisuus ja tarkkuus toimintavarmuutta. Tarkoitus on analysoida kaikki vaatimukset, mitä kaventavat resurssit ja mahdolliset aikaisemmat analysoinnit. Yhdelle vaatimukselle voi päteä yksi tai useampi laatuominaisuus ja avainlause. Taulukko perustuu lähteen [Lawrence & Gallager 1997] avainsanoihin.

Avainlauseet ovat negatiivisia ilmauksia: tietty virhetoiminto ei saa tapahtua. Niiden lisäksi analyysissa tulisi tarkastella myös positiivisilla ilmauksilla eli hakea ei-toivottuja vaikutuksia silloin, kun tietty vaatimus toteutuu. Avainlauseista ei välttämättä tarvitse käsitellä kaikkia ominaisuuksia, mutta analyysin kattavuus on sitä täydellisempi mitä useampia avainlauseita käytetään. Lisäksi monet muut laatukriteerit, kuten ristiriitaisuudet, virheellisyydet ja monikäsitteisyydet, tulevat tarkemmin käsiteltyä. Tosin viimeiksi mainittujen kriteerien tarkasteluun sopivat myös mm. oikeellisuuden todentamismenetelmät, joita esitettiin luvussa 5.

Analysointitavalle on myös tunnusomaista, ettei tarvitse selvittää täsmällisesti, mistä tietty tarkasteltava tila on peräisin tai onko tila yleensä mahdollinen edes saavuttaa, vaan analyyseissa yksinkertaisesti oletetaan, että tilan tai tapahtuman ehdot ovat voimassa, ja mahdollisia seurauksia tarkastellaan siltä pohjalta.

7.3 Analyysin suunnittelu

Stérna-analysointi on syytä suunnitella etukäteen huolellisesti, sillä epätäsmällisellä analysoinnilla kulutetaan helposti ylimääräistä työaikaa tulosten jäädessä silti laihoiksi. Päämäärä täytyy tiedostaa selkeästi. Se on tärkein analyysin suunnittelun tarvitsema tieto. Sitä palvelevat muut analysoinnissa tarvittavat perustiedot, joista joidenkin hankkiminen vaatii myös ennakolta tehtävää työtä.

Aina ei ole selvää, että juuri priorisoiminen on oikea tapa päämäärän saavuttamiseksi. Myös siksi tavoitetarkastelu on merkittävää. Testit, staattiset analyysit ja katselmukset voivat tietyissä tapauksissa olla täysin riittäviä. Keskeisimpänä tavoitteena Stérna-analyysin käyttöönotolle on tarve varmistua siitä, että ohjelmisto ei aiheuta kriittisiä seurauksia toimintavarmuuden, käytettävyyden, ylläpidettävyyden ja turvallisuuden kannalta. Tämän säännön tulee toteutua kaikissa tapauksissa, muita tavoitteita on kustannustehokkuus ohjelmistokehityksen elinkaaren prosessien ohjauksessa sekä yrityksen ohjelmistotuotannon kehittämisessä suuntaamalla laadunvarmistuksen toimenpiteitä automatisoituihin tarkistustekniikoihin.

Stérna-menetelmän ensimmäinen tarkastelu kohdistuu ohjelmiston toiminnalliseen vaatimusmäärittelyyn. Sitä ennen ovat etenkin sulautettujen ohjelmistojen kehityksessä yleensä ollut käyttäjävaatimusten ja järjestelmävaatimusten määrittelyt, jotka suuntaavat ohjelmistoprosesseja sopivalle luotettavuusprofiilille.

Alustavissa tehtävissä laaditaan suunnitelma, jossa mm. täsmennetään analyysitekniikoiden valintakriteerit, laaditaan analyysien apuvälineet ja valitaan varsinaiset analysoijat ja -ryhmät. Yhteenvetona suunnitelma sisältää siis seuraavat alustavat tehtävät:

1) Määritetään tarkasteltava kohde ympäristöineen.
2) Analysoidaan järjestelmätasot ja laaditaan kriittisyyslista.
3) Määritetään luotettavuusprofiili.
4) Laaditaan suunnitelma.

7.3.1 Tarkastelukohde

Ennen varsinaisen analysoinnin aloittamista rajataan tarkasteltava kohde sekä tarkastellaan, mitä tuloksia tarkastelusta halutaan ja mitä tuloksia on jo olemassa aikaisemmista analyyseista sekä millaisilla välineillä analysoinnit tehdään. Kohde määritellään mahdollisimman varhain, jotta ensimmäisellä analysoinnilla olisi jo ohjaava vaikutus myöhempiin analyyseihin, testeihin sekä ohjelmistoprosesseihin virheitten ja mahdollisten poikkeamien seurausten tunnistamisessa.

Tarkasteltavan kohteen rajaus sisältää seuraavia asioita:

- ohjattavan järjestelmän yleiskuvauksen esittäminen
- rajojen ja liittymien määrittelyt muihin järjestelmiin, sekä inhimilliset, fyysiset että toiminnalliset rajat
- ympäristön määrittely
- toimintaolosuhteiden ja kaikkien tarkoituksenmukaisten rajoitusten määrittely.

Analyysitarve johtaa lähtötavoitteiden määrittämiseen. Lähtötavoitteet ovat ylimmän tason kuvauksia siitä, miten luotettavasti ja turvallisesti järjestelmän ja siten myös ohjelmiston tulisi toimia. Niiden pohjalta kyetään arvioimaan, mitkä tapahtumat ovat niin merkittäviä uhkia tai vaaroja, että jatkotoimet ovat tarpeen.

7.3.2 Järjestelmätasojen vaara-analyysit ja kriittisyyslista

Seuraavaksi laaditaan kriittisyyslista, jossa esitetään ja kuvataan mahdolliset vakavat poikkeamien vaikutukset järjestelmätasoilla. Järjestelmätasot voivat olla esimerkiksi ohjattava prosessi, säätöventtiili tai jokin muu, johon sovellusprojektin ohjelmistolla on ohjaavaa vaikutusta. Kriittisyyslistaan kirjattavia vaikutuksia ovat mm. prosessin vaarat (räjähdys, kalliiksi tuleva keskeytys jne.), jotka usein on selvitetty poikkeamatarkastelun tyyppisillä riskianalyysitekniikoilla.

Kriittisyyslistan laatimisella tarkoitetaan ohjelmiston toiminnallisen tason vaatimusmäärittelyä, ylempien järjestelmätasojen vaarojen tunnistamista ja alustavaa seurausten merkityksen arviointia [MIL-STD-882B 1994, Friedman & Voas 1997]. Yksi tapa tunnistamiseen on haastatella järjestelmätasojen asiantuntijoita. Ainakin osa vaaroista saadaan mahdollisista aikaisemmin tehdyistä vaara-analyyseista, käyttökokemuksista tai mahdollisista onnettomuuksista. Tiedostamattomien vaarojen tunnistamiseksi tulisi käyttää tietyt tilanteet kattavia riskianalyysin vika- ja vaikutusanalyyseja.

Vika- ja vaikutusanalyyseihin liitetään mukaan riskin suuruuden arviointi, jolla tulisi tutkia tunnistettujen vaarojen alkutapahtumat tai olosuhteet, kyseessä olevat tapahtumaketjut, riskin vähentämismahdollisuudet ja mahdollisten vahingollisten seurausten luonne ja taajuus, jotta saataisiin kattava mitta analysoitavien riskien tasolle. Mitta voi sisältää ihmisiin, omaisuuteen tai ympäristöön kohdistuvat riskit, ja sen tulisi sisältää arvio arviointeihin liittyvästä epävarmuudesta.

Vaarojen tunnistamisessa voidaan käyttää apuna seuraavia dokumentteja tai konsultoida asiaa tietäviä:

- luonnossuunnitelmat ja -piirrokset
- erilaiset toimintakaaviot
- alustavat valmistus-, testi-, ylläpito-, korjaus- ja käyttösuunnitelmat
- muut analyysit, testit ja hallinnolliset toimet.

Leveson [1995] luettelee hyödyllisiä apukeinoja (ks. Taulukko 17) kriittisyyslistan laatimiseksi mutta mainitsee erityisesti, että vaarojen tunnistamiseen ei ole olemassa hyviä ja jäsenneltyjä menettelytapoja.

Taulukko 17. Levesonin [1995] suosittamat toimenpiteet vaarojen tunnistamiseksi.

Ohje	Tarkenne
Tarkastele olemassa olevia tietoja.	Historiatietoja, häiriö- ja vikaraportteja, onnettomuustietoja.
Hyödynnä julkaistuja tietoja vaaroista.	Vaarojen tarkistuslistat, standardit ja suunnitteluohjeet, jotka usein peilautuvat tapahtuneista onnettomuuksista.
Tarkastele järjestelmän energiaa.	Energialähteitä, -virtoja ja -kohteita.
Tarkastele vaarallisia aineita.	Polttoaineet, ponnekaasut, laserit, räjähdysaineet, myrkylliset ja paineistetut aineet.
Tarkastele aikaisemmin tehtyjä vaara-analyyseja.	Järjestelmätasojen HAZOP, FMECA ja ETA.
Tarkasta tehtävä- ja suoritusvaatimukset käyttöympäristössä.	Liitä tarkasteluun kaikki mahdolliset järjestelmän käyttäjät, ympäristöt, käyttötilat ja -ajat.
Ota huomioon rasitustilat.	Eri toimintatilat.
Tarkastele käyttöliittymiä.	Operaattorin ja automaattisen järjestelmän välinen vuorovaikutus.
Tarkastele erityisesti muutostilanteita järjestelmässä, teknisessä ja sosiaalisessa ympäristössä sekä järjestelmän tilanvaihdoissa.	Onnettomuusalttiita kohteita ovat käynnistys, uudelleen käynnistys, alasajo, testaus, uusien menetelmien kokeilu, huolto, korjaus, tarkastus, häiriön etsintä, modifikaatiot, stressitilanteet jne.
Käy läpi koko prosessi kohta kohdalta.	Mikä voi mennä vikaan, kuinka korjata ja mitä tehdä, jos niin tapahtuu.

Kriittisyyslistassa vaarat luetteloidaan määrättyyn formaattiin, mikä täydellisenä voi sisältää kaikki seuraavat Levesonin [1995] suosittelemat tiedot:

- osa tai järjestelmä tai yksikkö – se ryhmä, jossa vaara on
- vaaran kuvaus
- vaaran syy
- mahdolliset vaikutukset järjestelmätasolla ja ympäristöön
- vaaran taso
- korjaavat ja ehkäisevät keinot, suojaukset, suositukset ja suunnitteluohjeet
- käyttötila

- vastuuorganisaatio toimenpiteille
- todentamismenetelmät (testit, demonstraatiot, analyysit, tarkistukset) vaaran tehokkaasta kontrollista
- muut ehdotukset ja välttämättömät toimet.

Kriittisyyslistaa voidaan myös hyödyntää järjestelmätasoilla mm. kehitettäessä ja laadittaessa seuraavia tehtäviä ja asioita:

- järjestelmätason luotettavuusvaatimukset, -ohjeet ja -kriteerit
- suoritus- ja suunnitteluspesifikaatiot
- järjestelmäsuunnittelun arvioinnit
- testisuunnitelmat
- käyttöohjeiden valmistelu
- hallinnollinen suunnittelu.

Järjestelmätasot ovat hierarkkinen rakenne, joka tarkentuu sovellusohjelmiston toimintoihin. Ylin taso on ohjelmistolla ohjattava prosessi, järjestelmä, laite tai laitos. Järjestelmätasojen analyysit käsittävät kaikki ne mahdolliset tasot, jotka ovat ohjelmiston vaatimusmäärittelyä ylempänä. Tasoja voi olla useampia (ohjelmiston sisältävä komponentti, laite, järjestelmä), joita ei ole vielä käsitelty alustavan kriittisyyslistan tekemisessä.

Tarkastelu on aluksi laitteistokeskeistä (toimielimet, anturit, …) ja suuntautuu myöhemmissä analyyseissa järjestelmätoimintoihin [Leveson 1995]. Tunnistettuihin vaaroihin liitetään myös syiden tunnistaminen ja ehdotetaan ohjaavia tai suojaavia toimenpiteitä., mikä merkitsee sitä, että soveliaimpia tekniikoita olisivat vika- ja vaikutusanalyysien tyyppiset tekniikat. Järjestelmätason analyysi tarkastelee yksityiskohtaisesti mahdollisia vaaroja, jotka aiheutuvat osajärjestelmien rajapinnoista tai hajautetun järjestelmän toiminnasta kokonaisuudessaan. Leveson [1995] käynnistää järjestelmän vaara-analyysit järjestelmäsuunnittelun alussa ja jatkaa, kunnes suunnitelma on valmis. Hän suosittelee, että analyysit suoritetaan suunnittelukatselmusten yhteydessä.

Osajärjestelmien vaara-analyysien [MIL-STD-882B 1994, Leveson 1995] tarkoituksena on paljastaa vaarat, jotka liittyvät osajärjestelmien suunnitteluun, komponenttien vikaantumiseen, inhimillisiin virheisiin ihminen-kone-yhteydenpidossa tai osajärjestelmien toiminnallisiin suhteisiin. Analyysi käynnistyy, kun osajärjestelmät on suunniteltu riittävän yksityiskohtaisesti.

Alustavan kriittisyyslistan kattavuuden varmistamiseksi tarkastellaan ja arvioidaan, minkälaisia vaaroja ohjelmistolla voi olla järjestelmätasolla. Ohjelmistolla on ainakin seuraavat neljä mahdollista vaikutustapaa kuhunkin kriittiseen luotettavuustekijään [Lawrence & Gallager 1997]:

1. Ohjelmiston virhetoiminta voi merkitä kriittistä tilaa järjestelmätasoilla, mikä edellyttää joko kriittisen tilan eliminoimista tai vaikutuksen lieventämistä jollakin toimenpiteellä (esim. varmistava järjestelmä).
2. Ohjelmiston tehtävänä voi olla estää kriittisen tapahtuman syntyminen. Siten ohjelmiston virhetoiminta voi aiheuttaa kriittisen tapahtuman johtamisen ei-toivottuun seuraukseen, mahdollisesti onnettomuuteen.
3. Ohjelmiston tehtävänä voi olla siirtää järjestelmän tila kriittisestä vähemmän kriittiseen.
4. Ohjelmistoa voidaan käyttää seurausvaikutusten lieventämiseen.

Määritellään jokaisen tunnistettavan vaaran riskitaso (vakavuusluokka ja esiintymistodennäköisyys) käyttäen kohdassa 3.5.2 kuvattua taulukointiin perustuvaa menetelmää.

7.3.3 Luotettavuusprofiili

Luotettavuusprofiili (ks. luku 3) kuvaa luotettavuustavoitteet, joiden toteutumista Stérnalla tarkastellaan. Se määritetään yleensä suoraan järjestelmätasojen vaara-analysoinneista ja merkitään kriittisyyslistan kuhunkin vaaraan. Usein se kuitenkin laaditaan iteroivasti ensimmäisessä Stérna-analysoinnissa. Ohjelmiston elinkaarivaiheissa luotettavuusprofiilia allokoidaan yksityiskohtaisiin ohjelmistokomponentteihin (Kuva 5). Allokoitaessa luotettavuustavoitteet voivat vähentyä tai myös kasvaa. Tavoitteena on aina pienentää luotettavuusprofiilia, esimerkiksi arkkitehtuurisuunnittelussa virhesietoisilla ratkaisuilla, mutta usein se juuri ohjelmiston erityisominaisuuksista johtuen saattaa kasvaa. Esimerkiksi tavoite voi kasvaa arkkitehtuurisuunnittelussa, jos usea sovellusohjelmiston toiminto kohdistuu samaan ohjelmiston arkkitehtuuriosaan, tai suunnittelussa ja koodauksessa, jos käytetään riskialttiita menetelmiä, esimerkiksi muistin dynaamista hallintaa tai standardoimattomia ohjelmakieliä ja -kääntäjiä.

7.3.4 Luotettavuusanalyysin suunnitelma

Laaditaan suunnitelma, jossa täsmennetään tarkastelun tavoitteet, valitaan työkalut ja apuvälineet sekä laaditaan projektisuunnitelma.

Stérnaan kohdistuvat *tavoitteet* keskittyvät luotettavuusvaatimusten kelpoistamiseen. Lopputuloksena analyysien suorittamisesta selviävät arviot ohjelmiston riittävästä luotettavuudesta ja ohjelmistoprosesseja ohjaavista painotuksista, jotka liittyvät pääasiassa testauksiin ja tarkastuksiin.

Luotettavuusvaatimukset kelpoistetaan sopivasti valituissa (ks. kohta 6.3) ohjelmistotuotannon elinkaaren vaiheissa. Kelpoistuksessa tarkastellaan ensi sijassa luotettavuusvaatimusten ja niiden toteuttamisen täydellisyyttä, virheettömyyttä ja ristiriidattomuutta. Luku 3 kuvaa, miten luotettavuusvaatimukset johtavat suoraan toimintojen tietylle kriittisyysasteelle (esimerkiksi luotettavuusprofiilin mukaisesti).

Tavoitteena voidaan asettaa yksityiskohtaisen vaaratapahtuman selvittäminen analysoitaessa. Lähtökohtina tähän ovat silloin kriittisyyslista ja etenkin järjestelmätasojen vaatimusmäärittelyt, joissa esitetään myös luotettavuusvaatimuksia, jotka kohdistuvat kriittisyyslistassa esitettäviin uhkiin.

Myöhemmin elinkaaren vaiheissa täsmennetyt ja lisätyt luotettavuusvaatimukset käsitellään vastaavassa Stérna-analyysissa ja tarpeen mukaan toiminnallista vaatimusmäärittelyä koskevassa Stérnassa.

Stérnan vaatimia *työkaluja ja apuvälineitä* ovat toimintamallit, johon kuuluvat erilaiset uhkien ja vaarojen merkittävyyttä täsmentävät taulukot ja raportointitavat, jotka olisi tarkoitus ennen varsinaiseen analyysiin ryhtymistä sovittaa sovellusprojektille. Sovellettavat taulukot yleisessä muodossa ovat seuraavat:

- vaaratapahtuman vakavuusluokat (ks. kohta 3.5.2)
- vaaratapahtuman esiintymisasteikko (ks. kohta 3.5.2)
- riskitasot (ks. kohta 3.5.2)
- avainlauseet (ks. kohta 7.2).

Tunnistetut järjestelmävaarat saattavat johtaa suunnittelukriteereiden määrittämiseen. Suunnittelukriteerit eivät ole vaatimuksia vaan yleisohjeita, joiden pohjalta yksityiskohtaiset vaatimukset syntyvät sovelluksiin.

7.4 Vaatimusmäärittelyn luotettavuusanalysointi

Ohjelmiston toiminnallisen vaatimusmäärittelyn dokumentissa vaatimukset jaotellaan ryhmiin, jotka keskittyvät ohjelmiston toimintojen lisäksi myös tiettyihin laatuominaisuuksiin (ks. luku 3), kuten luotettavuusattribuutteihin. Luotettavuusattribuutit liittyvät kaikkiin kriittisiin toimintoihin. Toimintojen kriittisyydet eli merkittävyydet toimintavarmuuden, käytettävyyden, ylläpidettävyyden tai turvallisuuden suhteen merkitään

olennaisiksi osiksi toimintoa, jotta kyettäisiin hallitsemaan toiminnon suunnittelua, toteutusta, todennusta ja kelpoistusta.

Vaatimusten määrittelyvaiheen luotettavuustarkastelun tehtävänä on tarkistaa, että luotettavuusvaatimukset ovat oikein, täydellisesti ja yhdenmukaisesti johdetut ylemmän tason vaatimusmäärittelyistä sekä tarkistaa, että uusia vaaroja tai toimintoja ei ole esitetty. Lisäksi yhtenä tehtävänä on ohjata luotettavuusvaatimusten toteuttamista myöhemmissä vaiheissa priorisoimalla toiminnot määrättyihin luotettavuustasoihin.

Luotettavuusattribuuttien lisäksi määrittely sisältää läheisesti niitä sivuavia laatuominaisuuksia, kuten tarkkuuden ja robustisuuden. Tarkkuutta pidetään usein toimintavarmuuteen kuuluvana, robustisuutta taas käyttövarmuuteen kuuluvana ominaisuutena.

Seuraava aineisto tarvitaan vaatimusten määrittelyvaiheen analyyseissa:

- mahdolliset kriittisyyslistat ja luotettavuusprofiilit
- muut mahdolliset vaara- tai luotettavuusanalyysien tulokset
- ohjelmiston vaatimusmäärittelyn dokumentaatio.

Vaatimusmäärittely analysoidaan yhdeksässä askeleessa seuraavalla tavalla:

1. Valitaan A) kriittisyyslistan pohjalta jatkotarkasteluun sopivat kriittiset vaatimukset, tai B) kaikki vaatimukset.
2. Tarkastetaan tunnistettujen vaatimusten oikeellisuus ja täydellisyys.
3. Tarkastellaan avainlauseilla jokaista tunnistettua vaatimusta.
4. Määrätään tunnistettujen vaatimusten vakavuusasteet.
5. Laaditaan vaatimuksille luotettavuusprofiili.
6. Suositetaan ja ohjeistetaan suunnittelu- ja testiprosesseja tai modifioidaan vaatimusmäärittelyä.
7. Valitaan arkkitehtuurisuunnittelun luotettavuusanalyysia varten tarkasteltavat vaatimukset ja tapahtumat askeleesta kolme.
8. Raportoidaan tulokset.
9. Käsitellään tulokset vaatimusmäärittelyn katselmuksissa.

Jos aikaisemmissa elinkaarivaiheissa on tunnistettu riittävän hyvin vaaratapahtumat, *ensimmäisessä askeleessa* tunnistetaan niitä vastaavat vaatimukset jatkokäsittelyä varten. Muussa tapauksessa jatkokäsittelyyn valitaan kaikki vaatimukset.

Toisessa askeleessa tarkistetaan, että ohjelmistovaatimukset ovat oikeita ja täydellisiä. Virheettömyys on tärkeä ominaisuus, koska oikean ja toimivan ohjelman aikaansaaminen on yksi ohjelmistotuotannon perustavoitteista. Tarkoituksena tässä askeleessa on kyetä arvioimaan, että kaikki vaaditut elementit ovat olemassa ja oikein toteutettu, mikä auttaa seuraavan elinkaarivaiheen toteuttajia.

Kolmannessa askeleessa tarkastellaan avainlauseilla kutakin vaatimusta. Avainlauseiden käytön tarkoitus on avata keskustelua ja ehdottaa tarkastelumahdollisuuksia, ei rajoittaa analysointia. Avainlauseita on useita, kaikki eivät sovellu kaikkiin vaatimuksiin.

Neljännessä askeleessa, esimerkiksi avainlausetarkastelun yhteydessä, määrätään vaatimusten riskitasot kohdassa 3.5.2 esitetyllä tavalla vaikutusluokkien ja esiintymistiheyksien pohjalta. Vakavuustasot kootaan *seuraavassa* askeleessa luotettavuusattribuuteittain luotettavuusprofiiliksi.

Kuudennessa askeleessa päätetään, mitä kriittisimmille vakavuusasteisille vaatimuksille pitäisi tehdä. Yhtenä kriteerinä voidaan pitää vaatimuksen todennettavuutta. Jos tietyin toimenpitein, esimerkiksi testaamalla, ei vaaratapahtuman esiintymistiheyttä saada osoitetuksi riittävän matalaksi, vaatimus tulisi konstruoida uudelleen tai lisätä kriittisyyttä vähentäviä toimintoja eli uusia vaatimuksia määrittelyyn.

Seitsemännessä askeleessa valitaan tai esitetään jatkoanalysoitavat kohteet. Valintakriteeri on vaatimuksittain määritetty vakavuustaso. Vakavuustaso voi olla ehdotettujen toimenpiteiden jälkeen määräytyvä taso. Tällainen toimenpide voi olla esimerkiksi riittäviksi todettavat testaukset.

Kahdeksannessa askeleessa tulokset raportoidaan myöhempiä analysointeja sekä päätelmiä varten. Analyysien tuloksena saadaan seuraavat asiat:

- luettelo ohjelmistovaaroista
- ohjelmiston toimintojen riskitaso
- allokoitu luotettavuusprofiili
- ohjelmistovaarojen vaikutukset sekä oikealla että virheellisellä toiminnalla.

Myös raportit ovat todistuskappaleita suoritetuista analyyseista. Ne hyödyntävät uusinta-analyyseissa, jos esimerkiksi käyttöönoton jälkeen joudutaan tekemään muutoksia kohteeseen. Raportin esitystapa riippuu myös analysointitekiikasta, joita on käytettävissä useita. Yleisimmät näistä ovat vika-, vaikutus- ja kriittisyysanalyysi sekä vikapuuanalyysi (ks. luku 5). Tapahtumapuuanalyysia voitaisiin myös käyttää hyödyntämällä avainlauseita huipputapahtumissa ja laajentamalla puuta seurausten tarkasteluun. Tekniikan valinta riippuu monista tekijöistä, joita on selostettu luvussa 6.

Askeleessa yhdeksän tulokset käsitellään vaatimusmäärittelyn katselmuksissa. Sekä analyysin tuloksilla avainlauseita käytettäessä että vakavuus- tai riskitasojen asettamisella voi olla huomattavaa arvoa ohjelmistotuotannon myöhemmissä vaiheissa määriteltäessä resurssitarvetta sekä suunnittelulle, todentamiselle että testaukselle. Tulosten pohjalta voidaan myös päätellä uudelleensuunnittelun tarpeesta ohjelmistovaikutteisten riskien vähentämisessä.

Analyysin tulokset voivat johtaa muutoksiin järjestelmätason suunnittelussa, esimerkiksi tuloksena voi olla, että systeemivaatimukset, joita ajateltiin toteutettavan ohjelmistolla, toteutetaankin riskitekijöistä johtuen laitteistolla. Toisaalta analyysin tulokset saattavat merkitä sitä, että tietyt vaatimukset voidaan katsoa vähemmän kriittisiksi eikä niitä kannata tarkastella jatkoanalyyseissa.

Yksi mahdollisuus tarkkailla erityisesti ohjelmiston luotettavuuden kehittymistä elinkaaren eri vaiheissa on eriyttää ohjelmistovirheiden esiintymistodennäköisyydet satunnaisvikojen esiintymistodennäköisyyksistä. Uusi indeksi ”havaintotaso” riippuu mm. testausten ja muiden todennus- ja kelpoistustapojen kattavuudesta sekä mahdollisten ohjelmistometriikoiden tuloksista. Projektin alussa vaatimusmäärittelyssä havaintotason indeksi on korkea alentuen sitä mukaan myöhemmissä analyyseissa, kun uusia todisteita virheettömyydestä ilmaantuu.

7.5 Arkkitehtuurisuunnittelun luotettavuusanalyysi

Tietokoneen arkkitehtuuri jakaantuu kolmeen osaan:

1. Laitteiston arkkitehtuurin, joka sisältää prosessorit, muistit, ajurit, näyttölaitteet ja tietoyhteydet.
2. Ohjelmiston arkkitehtuurin, joka sisältää ohjelmistoprosessit, muistit, näyttölayoutit ja tietokoneväylät.
3. Laitteisto-ohjelmisto-ihminen-integraatioon, joka kuvaa,
   - kuinka ohjelmisto toimii laitteistossa
   - mitkä prosessit toimivat missäkin prosessorissa
   - missä eri muistit sijaitsevat
   - missä eri näytöt esitetään
   - kuinka tietoliikenne fyysisesti tapahtuu
   - kuinka operointi tapahtuu ja mikä on ihmisen asema kokonaisuudessa.

Arkkitehtuurisuunnittelun luotettavuusanalyysi jatkaa edellisen analyysin tuloksista merkittävimmiksi todettujen tapahtumien käsittelyä ohjelmistoarkkitehtuurin osalta. Tapahtumat voivat liittyä vioittumistapojen syihin, havaintotapoihin tai suojaaviin keinoihin. Niistä edetään sopivalla esitystavalla, joko ohjelmiston FMECA- tai FTA-tekniikalla. Vastaavasti voidaan menetellä laitteiston osalta, mutta tässä keskitytään ohjelmistoon.

Arkkitehtuurin suunnitteluvaihe on erityisen tärkeää juuri luotettavuuden kannalta, koska monet luotettavuutta parantavat tai heikentävät ratkaisut tehdään juuri tässä vaiheessa. Luotettavuutta parantavat virhesietoiset suunnitteluratkaisut ja heikentää kriittisten vaatimusten tai toimintojen keskittyminen vain tiettyihin arkkitehtuuriosiin. Parannukset ja heikennykset vaikuttavat luotettavuusprofiiliin, joten se tulee muuttaa arkkitehtuurisuunnittelun mukaiseksi.

Luotettavuusprofiilia heikentävät arkkitehtuuriosiin keskittymisten lisäksi tietyille arkkitehtuureille ominaiset vioittumistavat, joita muilla arkkitehtuuriratkaisuilla ei ole. Nämä merkitsevät lisävaaroja yksityiskohtaisessa suunnittelussa, jos niihin ei ole osattu varautua.

Seuraava aineisto tarvitaan arkkitehtuurivaiheen Stérnassa:

- alustava kriittisyyslista
- krittiisyysprofiili
- edeltävien analyysien tulokset
- edeltävät tarkasteluaineistot (mm. ohjelmiston toiminnallinen vaatimusmäärittely)
- jäljitettävyysmatriisi vaatimuksista arkkitehtuuriosiin ja takaisin
- ohjelmiston arkkitehtuurikuvaus.

Arkkitehtuurisuunnittelun dokumenttien tulisi sisältää kaksisuuntainen jäljitettävyys vaatimusten ja arkkitehtuuriosien välillä. Jokaisesta vaatimuksesta tulisi olla osoitettavuus sen toteutuksesta aina koodiin asti ja toisaalta kaikista suunnittelu- ja toteutusratkaisusta osoitettavuus siihen, mihin ’käskyihin’ ratkaisut perustuvat. Jos osoitettavuus- eli jäljitettävyystietoa ei ole olemassa, analysointi vaikeutuu niin paljon, että jäljittävyys tulisi tehdä ennen kyseisten analyysien aloittamista.

Arkkitehtuurivaiheen analyysi alkaa edellisen analyysivaiheen tulosten tarkastelulla ja päättyy raportointiin ja jatkoanalyysien kohteiden valintaan:

1. Tunnistetaan edellisen analyysin tuloksista jatkotarkasteltaviksi valitut vaatimukset ja niitä vastaavat arkkitehtuuriosat.

2. Tarkastetaan tunnistettujen kriittisten arkkitehtuuriosien oikeellisuus ja täydellisyys.
3. Määrätään niiden luotettavuusprofiili.
4. Analysoidaan ne avainlauseilla.
5. Varmistaudutaan siitä, että kriittisiin vaatimuksiin kohdistuvien testiprosessien kattavuus on riittävä. Tarvittaessa suositetaan testaustoimenpiteitä.
6. Päätetään jatkoanalyysien tarpeesta.
7. Raportoidaan tulokset.
8. Käsitellään tulokset arkkitehtuurisuunnittelun katselmuksissa.

Ensimmäisessä askeleessa tunnistetaan edellisen luotettavuusanalyysin tuloksista kriittiset vaatimukset ja määritetään jäjitettävyysmatriisista niitä vastaavat arkkitehtuuriosat. Seuraavassa askeleessa tunnistetaan arkkitehtuuriosien laatukriteereistä ainakin oikeellisuudet ja täydellisyydet.

Kolmannessa askeleessa allokoidaan luotettavuusprofiili tunnistettuihin arkkitehtuuriosiin. Allokointiprosessi ei ole kovin yksinkertainen, sillä monet vaatimukset tai toiminnot kohdentuvat vain tietyille arkkitehtuuriosille. Samoin yksittäinen vaatimus voi vastaavasti jakaantua usealle arkkitehtuuriosalle. Tämä merkitsee myös sitä, että yksi arkkitehtuuriosa koostuu useasta vaatimuksesta, joille edellisessä vaiheessa on määritetty luotettavuusprofiilit. Määritystehtävään Lawrence & Gallager [1997 esittävät yksinkertaisen lähestymistavan, josta voisi laatia yritys- tai projektikohtaisesti sopivia malleja. Määritystapaa esittää taulukko 18.

Taulukko 18. Arkkitehtuuriosan riskitason määrääminen.

Arkkitehtuuriosan riskitaso	**Arkkitehtuuriosan riskitaso sen jälkeen, kun yksi uusi vaatimus on lisätty**		
	Korkea	**Keski**	**Matala**
Hyvin korkea	Hyvin korkea	Hyvin korkea	Hyvin korkea
Korkea	Hyvin korkea	Korkea	Korkea
Kohtalainen	Korkea	Korkea	Kohtalainen
Matala	Korkea	Kohtalainen	Matala

Taulukkoa 18 käytetään seuraavasti:

- Otetaan arkkitehtuuriosaa koskeva ensimmäinen (yksi) vaatimus. Määrätään arkkitehtuuriosan riskitaso samaksi kuin tämän vaatimuksen riskitaso.
- Käsitellään seuraavat vaatimukset yksi kerrallaan ja aina määrätään arkkitehtuuriosan uusi riskitaso taulukosta olettamalla, että kaikki vaatimukset epäonnistuvat samanaikaisesti.
- Jatketaan, kunnes kaikki arkkitehtuuriosaan vaikuttavat vaatimukset on käsitelty.

Neljännessä askeleessa hyödynnetään jälleen taulukoituja avainlauseita (taulukko 15) jokaiselle kriittiselle arkkitehtuuriosalle. Prosessin jälkeen seuraavassa askeleessa (*askel 5*) päätetään riskitason suuruuden perusteella jatkotoimista. Myöhemmissä elinkaarivaiheissa teknisillä ratkaisuilla ei saada riskitasoa enää alennettua, joten arkkitehtuurisuunnittelu on tässä mielessä merkittävä. Suunnittelu-, toteutus- ja testausvaiheissa voidaan alentaa riskitasoa vain toteamalla esiintymistiheys pieneksi.

Seuraavat johtopäätökset voidaan tehdä, jos jokin arkkitehtuuriosan riskitaso osoittautuu liian suureksi:

- Suunnitellaan uusi arkkitehtuuri.
- Alennetaan arkkitehtuuriin kohdistuvaa riskiä (mm. yksinkertaisuus, panostaa erityisesti todentamiseen ja kelpoistukseen, jne.).
- Käytetään korvaavia tekniikoita kokonaissovelluksen riskin alentamiseksi.
- Varmistutaan kriittisiin arkkitehtuuriosiin kohdistuvien testiprosessien kattavuudesta.

Myöhemmät kehitysvaiheiden panostaminen (myös todennukset ja testit) riippuvat erityisesti juuri arkkitehtuurivaiheen analyysien tuloksista. Analyysilla voidaan myös osoittaa, että tietty arkkitehtuuriosa ei ole kriittinen eikä se siten vaadi erityispainotusta.

Askeleessa kuusi valitaan suunnittelun luotettavuusanalyysissa käsiteltävät asiat.

Arkkitehtuurivaiheen analyysin tulosdokumentit ovat seuraavat (*askel 7*):

- luettelo kriittisistä arkkitehtuuriosista ja niiden riskitasoista
- ohjelmistovirheiden vaikutukset
- suunnittelu- ja koodaussuosituksia tai -rajoituksia
- riskin vähentäminen.
- suositukset lisäanalyyseille ja -testeille.

Tulokset käsitellään (*askel 8*) arkkitehtuurisuunnittelun katselmoinneissa. Tilaisuuksissa esitetään merkittävät arkkitehtuurin luotettavuusongelmat ja niihin kohdistuvat suositukset sekä päätetään jatkoanalyyseista ja muutostarpeesta.

7.6 Suunnittelun luotettavuusanalyysi

Suunnittelu luotettavuusanalyysin tuloksista selviää, ovatko systeemin riskit muuttuneet. Jos aikaisemmat luotettavuusanalyysit ovat hyvin tehtyjä, on myös oletettavaa, ettei uusia riskitekijöitä ilmaannu suunnittelussa. Suunnittelun luotettavuusanalyysin jälkeen voidaan varmistua siitä, ettei riskitekijöitä ole, ja voidaan siirtyä yleisestä tarkastelutavasta yksityiskohtaisempaan toteutuksen tarkastukseen. Lisäksi analyysitulosten perusteella voidaan painottaa resurssointia koodaukseen ja testaukseen.

Seuraavaa aineistoa tarvitaan suunnitteluvaiheen luotettavuusanalyysissa:

- alustava kriittisyyslista
- luotettavuusprofiili arkkitehtuurivaiheen jälkeen
- aikaisemmat luotettavuusanalyysin tulokset
- edeltävät tarkasteluaineistot (mm. ohjelmiston toiminnallinen vaatimusmäärittely, ohjelmiston arkkitehtuurikuvaukset)
- jäljitettävyysmatriisi arkkitehtuuriosista suunnitteluratkaisuihin ja takaisin
- ohjelmiston suunnitteludokumentaatio.

Suunnittelun luotettavuustarkasteluun esitetään seuraavanlaista askeleittain etenevää analysointitapaa:

1. Tunnistetaan jatkokäsittelyyn valittujen arkkitehtuuriosien suunnitteluosat.
2. Varmistaudutaan suunnittelun oikeellisuudesta ja täydellisyydestä edellisten analyysien tulosten edellyttämiltä osilta sekä mahdollisten erillisten suunnitteluun kohdistuvien suositusten osalta.
3. Analysoidaan avainlauseilla.
4. Varmistaudutaan siitä, että kriittisiin vaatimuksiin kohdistuvien testiprosessien kattavuus on riittävä.
5. Ohjeistetaan koodausta tämän ja edellisten analyysien tulosten pohjalta.
6. Päätetään jatkoanalyysien tarpeesta.
7. Laaditaan ohjeita mahdolliseen käyttöohjeeseen ja vastaaviin dokumentteihin.
8. Raportoidaan tulokset.
9. Käsitellään tulokset suunnittelukatselmuksissa.

Ensimmäisessä askeleessa tunnistetaan edellisessä luotettavuusanalyysissa tunnistettujen ja valittujen arkkitehtuuriosien vastaavat suunnitteluosat. Arkkitehtuuriosa voi koostua useasta suunnitteluosasta, ja suunnitteluosa voi pohjautua useampaan arkkitehtuuriosaan. Tällaisia osia ovat mm. laiteohjaimet. Varmistetaan, että valittuihin arkkitehtuuriosiin kohdistuvat vikatapahtumat ovat tiedossa kaikista edellisistä luotettavuustarkasteluista (esimerkiksi vikapuukaavioesityksenä).

Toisessa askeleessa varmistutaan siitä, että on suunniteltu ainakin laatukriteereiden oikeellisuuden ja täydellisyyden edellyttämällä tavalla.

Kolmannessa askeleessa tarkastellaan avainlauseilla (taulukko 19). Resurssien riittämättömyyden takia kannattaa tarkastella vain kriittisyysasteissa korkeimmat arkkitehtuuriosat tai ne suunnitteluosat, jotka ovat yhteisiä usealle arkkitehtuuriosalle (tietoliikennemoduulit, laiteohjaimet). Varsinaiseksi suoritustekniikaksi suositellaan ohjelmiston FTA-tekniikkaa, missä etsitään syitä avainlauseitten avulla. Myös ohjelmiston FMECA voi tulla kyseeseen, jos halutaan tunnistaa muita ei-toivottavia seurauksia kuin, mitä edellisessä analyysivaiheessa on tunnistettu.

Neljännessä askeleessa varmistutaan siitä, että kriittisiin suunnitteluosiin kohdistuvat testaukset ovat riittävän kattavia. Tarvittaessa suositetaan testauksia. Seuraavassa askeleessa (*askel 5*) ohjeistetaan koodausta, ja askeleessa kuusi suositetaan jatkoanalyysien tarpeesta. Valitaan toteutuksen luotettavuusanalyysin tarkasteltavat kohteet ja niissä käsiteltävät asiat (tapahtumat esimerkiksi vikapuukaaviona). *Askeleessa seitsemän* ohjeistetaan käyttäjää.

Suunnittelun luotettavuusanalyysin tulosdokumentit ovat seuraavat (*askel 8*):

- luettelo kriittisistä ohjelmistokomponenteista ja niiden riskitasoista
- mahdollisesti analysoidut ohjelmistovirheiden vaikutukset
- koodaussuositukset tai -rajoitukset
- suositukset lisäanalyyseille, mm. toteutuksen luotettavuusanalyysille ja testeille.

Suunnittelun luotettavuusanalyysin tulokset käsitellään suunnittelun katselmuksissa, joissa myös päätökset mahdollisista lisä- tai jatkotoimista tehdään.

Taulukko 19. Eräiden laatuattribuuttien avainlauseet suunnittelu- ja toteutusvaiheissa.

Laatu	Avainlauseet
Syötteen ja tulosteen tarkkuus	Jäävät nolliin tai ykkösiin (tai johonkin väliarvoon) Ovat alle tai yli minimirajan Ovat rajojen sisällä, mutta vääriä Fyysiset yksiköt ovat virheellisiä Väärä datatyyppi tai -koko
Laskentatarkkuus	Tulokset hyväksyttyjen virherajojen ulkopuolella Yhtälö tai kaava on väärä Fysikaaliset yksiköt ovat virheellisiä Väärä datatyyppi tai -koko
Viestien suorituskyky	Viestien määrä on alle tai yli tietyn määrän Viestien määrä on epävakaa Viestien sisältö on väärää, mutta uskottavan näköistä
Ajastus	Ei tule syötteitä tai ei lähde tulosteita Syötteet tai tulosteet ovat liian aikaisin tai liian myöhään Syötteet tai tulosteet ovat odottamattomia Koodi juuttuu päättymättömään silmukkaan Lukkiuma

7.7 Toteutuksen luotettavuusanalyysi

Deduktiivinen tapahtumaketju, jonka lähtökohtana ovat vaatimusten määrittelyssä tunnistetut mahdolliset virhetoiminnot ja joka täydentyy arkkitehtuurisuunnittelussa ja detaljisuunnittelussa, päättyy toteutuksen tarkastelussa. Kaikissa edeltävissä vaiheissa, kuten myös toteutuksen analyysivaiheessa, luotettavuusanalyysi kelpoistaa ohjelmiston luotettavuusvaatimukset, mitä osaltaan tukevat erillisenä toimenpiteenä myös oikeellisuuden todennukset. Aikaisemmissa analyyseissa kelpoistamistoimen ohessa pääpaino on ollut ohjelmistoprosessien ja analyysien ohjaamisessa. Ohjaava vaikutus on ollut suurimmillaan määrittelyn analyysivaiheessa ja vähentyy asteittain jälkimmäisissä vaiheissa. Toteutuksen analyysivaiheen tuloksena suositellaan etenkin dynaamisia tai rakenteellisia testejä sekä tietyin edellytyksin palaamista edeltäviin luotettavuusanalyyseihin.

Toteutuksessa tarkasteluiden oikeellisuuden todentaminen on tärkeämpää kuin mahdollisten uusien vaaratekijöiden tarkastelu, jos aikaisemmat kolme luotettavuusanalyysia on tehty riittävän perusteellisesti.

Yleiskäyttöiset riskianalyysitekniikat ohjelmiston FTA ja FMECA soveltuvat myös lähdekielisen koodin analysointiin. Erityisesti Leveson [& Harvey 1983, 1991, 1995] on kehittänyt vikapuutekniikkaa, joka perustuu eräiden rakenteellisten ohjelmakielten, nimittäin Pascalin ja Adan, käskykohtaisille mini-vikapuille (ks. luku 5). Muille kuin rakenteellisille ohjelmakielille, esimerkiksi C-kielelle, menetelmää ei ole tiettävästi sovellettu mini-vikapuiden kehittämishankaluuden takia. Myös ohjelmiston FMECA ei ole erityisessä koodintarkastajien suosiossa. Vaikka tiedettäisiinkin erityiset tarkastelukohdat ohjelmassa, seurausvaikutusten sekä havainto- ja estomekanismien tunnistaminen on työlästä, sillä tarkastelutaso on syvällä koodin sisällä, josta on lähes mahdotonta nähdä ohjelmistoa kokonaisuutena. Näitä kahta tekniikkaa suositeltavampia ovat staattiset analyysit, mm. saavutettavuusanalysaattorit ja koodintarkastukset.

Tässä toteutusvaiheen luotettavuusanalyysi jatkaa pääpiirteissään myös samaa menettelyä, jota aikaisemmatkin analysoinnit noudattavat. Tavoitteena on ollut kehittää menettelytapa, joka karsii ohjelmiston FMECA- ja FTA-tekniikoiden työläät ominaisuudet ja yhdistää menettelyn ohjelmistotuotannon tarkastus- ja katselmointiprosesseihin. Samaa edellisten analyysien kanssa ovat jossakin määrin avainlauseiden sekä mahdollisten uusien ominaisuuksien ja niiden seurausvaikutusten tunnistamiset. Lisäksi, samoin kuin aikaisemmissakin analyyseissa, estimoidaan tapahtumaketjun tietyn kohdan riskinsuuruus toteutusvaiheen jälkeen.

Toteutusvaiheessa luotettavuusanalyysi eroaa edellisistä lähinnä siinä, että pääpaino ei ole avainlauseprosessilla, vaan analyysi noudattaa tarkistuslistamenettelyä (luku 5) tai yhdistyy suoraan lähdekielisen koodin tarkistuksiin. Tarkistuslistat koostuvat mm. ohjelmakielen toteutussäännöistä. Lisäpiirteitä tarkistuksiin tuovat luotettavuusanalyysien aikaisempien vaiheiden tulosten hyödyntämiset. Niiden ja jäljitettävyysmatriisin perusteella tiedetään tarkasti, mitä kohtia lähdekoodista tarkastellaan, sekä lisäksi, mitä asioita niissä käydään läpi. Koko vikatapahtumaketju ylhäältä seurauksista määrittelyn, arkkitehtuurin ja suunnittelun kautta fokusoituu tiettyihin proseduureihin, funktioihin, aliohjelmiin tai yksittäisiin käskyihin tai niiden tiettyihin kokonaisuuksiin lähdekielisessä ohjelmakoodissa.

Aikaisempien luotettavuusanalyysien tulokset viestivät kriittiset tapahtumaketjut lähtien kunkin ketjun loppupäästä eli seurauksista. Alkupään muodostavat alkutapahtumat, jotka tapahtumaketju kokoaa pitkin ohjelmistotuotannon elinkaarivaiheita eri luotettavuusanalyyseissa. Ohjelmistovirheet ovat ensi sijassa suunnitteluvirheitä sisältyen lopulliseen ohjelmakoodiin. Koska niiden syntylähde voi olla esimerkiksi vaatimusten määrit-

telyssä, niitä ei välttämättä voida tunnistaa ohjelmakoodista. Määrittelyvirheitä ei myöskään aina kyetä tunnistamaan määrittelyvaiheen aikana, vaan myöhemmistä tarkastelujaksoista tai testauksista saadaan viitteitä mahdollisista määrittelyvirheistä.

Alkutapahtumat eivät kuitenkaan koostu virheistä, jotka on koottu elinkaaren vaiheista, sillä todetut todelliset virheet poistetaan sitä mukaa, kun ne on tunnistettu. Alkutapahtumat koostuvat mahdollisista virheistä, joiden olemassaolo on tarkoitus tarkistaa joko jo heti kyseisen tarkasteluvaiheen aikana tai myöhemmissä testauksissa ja tarkistuksissa.

Toteutusvaiheen luotettavuusanalyysiin kohdistuvat siis aikaisempien luotettavuusanalyysien tulokset, joiden kriittisyystasot ilmaistaan luotettavuusprofiililla. Profiilissa ovat mukana kaikki käsiteltävät luotettavuusattribuutit, ja ne myös kohdentavat toteutuksen tarkastelua mm. sopivien avainlauseiden valitsemiseksi. Kuten edellä jo todettiin, luotettavuusanalyysi perustuu toteutusvaiheessakin avainlauseiden käyttöön, mutta nämä ovat nyt hyvin sovelluskohtaisia. Normaalit tarkistuslistat ja ohjelmointisäännöt ovat myös hyviä perusteita vioittumistapojen valinnoille.

Seuraavaa aineisto tarvitaan toteutusvaiheen luotettavuusanalyysissa:

- alustava kriittisyyslista
- luotettavuusprofiili suunnitteluvaiheen jälkeen
- kaikki edeltävät luotettavuusanalyysin tulokset
- edeltävät luotettavuusanalyysien tarkasteluaineistot (mm. ohjelmiston toiminnallinen vaatimusmäärittely, ohjelmiston arkkitehtuuri- ja moduulisuunnittelukuvaukset)
- jäljitettävyysmatriisi suunnittelukuvauksista lähdeohjelman koodikohtiin ja takaisin suunnitteluun
- lähdekielinen koodi.

Toteutusvaiheen Stérna-tulosdokumentaatio sisältää analyysien tulokset.

Toteutuksen luotettavuusanalyysi noudattaa seuraavia askeleita:

1. Tunnistetaan tarkasteltavat kohdat ja asiat lähdekielisestä ohjelmakoodista edellisten analyysien tuloksista.
2. Tarkistetaan tunnistettujen ohjelmakoodin osien oikeellisuus ja täydellisyys.
3. Tunnistetaan mahdollisia virhetilanteita avainlauseiden avulla (taulukko 19).
4. Tarkastellaan mahdollisten yhteisvikojen esiintymistä.
5. Varmistaudutaan siitä, että lähdekoodi on riittävästi dokumentoitu.

6. Varmistutaan testien riittävyydestä, sekä suositellaan lisätarkasteluja.
7. Varmistaudutaan siitä, että kriittisten vaatimusten testikattavuus on riittävä.
8. Päivitetään käyttöoppaan ja muiden asianomaisten dokumenttien tietoja.
9. Dokumentoidaan analyysin tulokset.
10. Käsitellään analyysivaiheen tulokset vastaavissa toteutusvaiheen katselmuksissa.

Ensimmäisessä tunnistetaan ja valitaan edellisen analyysin tuloksista tarkasteltavat koodinosat ja tapahtumat, jotka seuraavassa vaiheessa tarkistetaan. Suositeltavin tekniikka on yksinkertainen kvalitatiivinen ohjelmiston vikapuuanalyysi (ks. luku 5). Mahdollisesti suunnittelun luotettavuusanalyysia joudutaan päivittämään tai löydetään uusia riskejä, jolloin mahdollisesti uusitaan kaikki edelliset analyysit asiaankuuluvilta osilta. Jos resurssit ovat riittämättömiä, kannattaa keskittyä joko riskitasoltaan korkeimpiin osiin tai perussysteemin toimintoihin.

Toisessa askeleessa varmistutaan siitä, että lähdekielinen ohjelma on koodattu ainakin laatukriteereiden oikeellisuuden ja täydellisyyden edellyttämällä tavalla edellisten luotettavuuden analyysitulosten ja -suositusten perusteella.

Kolmannessa askeleessa ohjelmaa tarkastellaan avainsanoilla esimerkiksi taulukon 19 tukemana. Tunnistetaan mahdolliset kriittiset tilat, jotka johtuvat syötteet/tulosteet-ajastuksesta, lukkiutumista, vääristä tapahtumista tai laitteiston vikaherkkyydestä.

Neljännessä askeleessa keskitytään yhteisvikatarkasteluun tunnistamalla riskialttiita ohjelmointisääntöjä sekä työkalujen käyttösääntöjä ja käyttöjärjestelmistä riskitekijät, mm. riskitoiminnot. *Seuraavassa askeleessa* tarkistetaan dokumentoinnin riittävyys ja ymmärrettävyys.

Kuudennessa askeleessa tarkistetaan, että kriittisten kohtien rakenteelliset ja toiminnalliset testit ovat riittäviä. Suositetaan tarpeen vaatiessa testitapauksia ja lisätarkastuksia jollakin soveliaalla tarkastustekniikalla (luku 5). *Askeleessa seitsemän* suositetaan käyttöoppaaseen täydentäviä ohjeita kriittisten vaatimusten osalta. *Kahdeksannessa askeleessa* tulokset dokumentoidaan.

Tarkasteluiden tuloksena *(askel 9)* voidaan päätyä lisätarkasteluiden suosittamiseen esimerkiksi siksi, etteivät käytetyt tekniikat ole riittäviä tietyssä kriittisyystasossa tai tarkastelukohde on niin kompleksinen, että tarvitaan kohdennettuja black- tai white-box-testejä siitä selviytymiseen.

8. Yhteenveto ja johtopäätökset

Luotettavuudesta puhutaan paljon. Se liitetään kaikkiin laatuominaisuuksiin, mutta sitä ei käsitellä käytännön ohjelmistoprojekteissa. Luotettavuusteoriaan ja lukuisiin käytännönläheisiin menetelmiin ja tekniikoihin tutustumiskynnys on usein korkea, ja perehtymisen puute estää merkittävien luotettavuustarkasteluiden käyttöönoton.

Julkaisussa kuvataan ohjelmiston luotettavuusvaatimusten kelpoistusmenetelmä, jossa on sovellettu riskianalyysin perustekniikoita. Menetelmä kehitettiin käytännön ohjelmistoprojekteissa koetarkasteluna useammassa vaiheessa. Menetelmän teoreettinen tarkastelupohja liittyi luotettavuustekniikan ohjelmistosovelluksiin, joissa käsiteltiin määrittely- ja suunnitteluvirheen kehittymistä ohjelmiston virhetoiminnoksi ja järjestelmän vikaantumiseksi. Luotiin lyhyt katsaus virheitä, virhetilanteita ja virhetoimintoja eliminoiviin, sietäviin ja ennustaviin menetelmin sekä luotettavuusattribuutteihin, joista puhutaan, koska ne liittyvät kriittiseen ohjelmiston hyödyntämiseen.

Virheiden ja virhetilanteiden käsittely kasvaa jyrkästi ohjelmiston kehittämistyön kuluessa. Virheiden korjaaminen ja oikeiden vaatimusten asettaminen mahdollisimman aikaisessa vaiheessa alentavat kustannuksia. Kuitenkaan kaikkia virheitä ei kyetä eliminoimaan alkuvaiheissa. Pyrkimys kehittää täydelliset ja oikeat vaatimukset sekä virheetön ohjelmakoodi ei siksi riitä, vaan lisäksi on rakennettava virhesietoisuutta, joka havaitsee ja korvaa virhetilanteet automaattisesti suorituksen aikana. Virhesietoisuuden rakentaminen myös maksaa ja monimutkaistaa järjestelmää, millä saattaa olla myös kielteisiä seurauksia. Korkeilla luotettavuuden vaatimustasoilla tulisi pyrkiä järjestelmätason virhesietoisuuteen siten, että ohjelmistosta aiheutuvaa riskiä vähennetään laitteistopohjaisilla tekniikoilla. Ohjelmistoon ei saa liiaksi luottaa.

Useimmiten kielteiset seuraukset tapahtuvat silloin, kun ohjelmisto ei täytä alkuperäisiä tarpeita, joita mahdolliset käyttäjät tai asiakkaat ovat asettaneet. Tarpeita joko ei ole kirjattu vaatimusmäärittelyihin tai niiden toteuttaminen ei ole onnistunut ohjelmiston kehityksen aikana. Jälkimmäiseen vaikuttavat kesken kehitystyön tehdyt vaatimusten muutostyöt ja uusien vaatimusten määrittelyt. Etenkin viime hetken pienet muutokset jäävät viimeistelyä vaille. Myös käyttäjän oletetaan voivan muuttaa tai lisätä uusia vaatimuksia. Ongelmia alkaa esiintyä, kun muutettuja tai uusia vaatimuksia ei jäljitetä tai projektin aikataulua täsmennetä. Luotettavuusanalyysien avulla jäljitetään virheitä ja virhetilanteita, mutta ilman täsmällistä mm. muutosten jälkeistä jäljittämisen hallintaa analyysi ei voi onnistua. Vain täsmällisen jäljitettävyyden, esimerkiksi jäljitettävyysmatriisin, avulla tiedetään, miten muutokset vaikuttavat vikamekanismissa. Tähän täytyy löytyä myös varauksia projektin aikataulutuksessa.

Kehitetyn ohjelmiston luotettavuusvaatimusten kelpoistusmenetelmän eräs hyöty on siinä, että ohjelmiston kehittäjät oppivat ymmärtämään järjestelmällistä ohjelmistovirheiden riskienhallintaprosessia. Mitkä vaikutukset ohjelmiston virhetilanteella on? Miten virhetilanne voidaan havainnoida? Mitä riskin vähimmäistämiskeinoja on käytettävissä virhetilanteen etenemisen estämiseksi?

Kelpoistusmenetelmä pitää ohjelmistoprojektin myös kurissa. Liian kompleksiset suunnitelmat ovat usein syynä epäonnistumisiin. Jos on määritelty ohjelmisto, jonka toteuttamiseen eivät mahdollisesti taidot riitä, arkkitehtuuri on puutteellinen, olio- ja tietokantasuunnittelut virheellisiä, koodi tehotonta, jne., menetelmän avulla voidaan jo varhaisessa vaiheessa todeta virhemekanismin kasvavan liian laajaksi.

Jäljittämisvaateen lisäksi menetelmän koekäyttö osoitti arkkitehtuuri- ja suunnitteluvaiheiden tärkeyden nimenomaan luotettavan ohjelmiston rakentamisessa. Usein niistä jompikumpi jätetään kokonaan pois ohjelmistoa kehitettäessä tai dokumentoidaan vasta ohjelman syntymisen jälkeen. Monet luotettavuuskriittiset päätökset tehdään näissä vaiheissa, ja systemaattisella kelpoistuksella täsmennetään päätösten oikeellisuutta.

Vaikka luotettavuudessa onkin aina kyse mittaamisesta, voi sekin olla laadullista, etenkin kun ohjelmiston luotettavuuden kvantitatiivnen mittaaminen on vain vaivoin toteutettavissa. Ohjelmiston kvantitatiivinen luotettavuusarvio antaa täsmällisemmät arvot kuin kvalitatiivinen, mutta jälkimmäisen etuna on kuitenkin pienen resurssitarpeen johdosta nopeus käsitellä luotettavuusaiheita. Menetelmä perustuu mahdollisten virhetoimintojen priorisointiin, mikä on aina suhteellisen helppoa ja virheetöntä.

Kelpoistamisen onnistuminen riippuu lähtödokumenttien saatavuudesta. Jos saatavuudessa, täydellisyydessä tai virheettömyydessä on puutteita, analyysiprosessi keskeytyy tai viivästyy tai jollakin muulla tavoin aiheuttaa ylimääräisiä analyysikustannuksia. Menetelmän suunnittelussa tulisi ottaa huomioon seuraavia asioita:

- aikavaraukset ongelmien käsittelyyn
- asiantuntijoiden tarve ja saatavuus
- lähtödokumenttien valmiusaste (ohjelmistotuotannon dokumentit)
- aikaisempien analyysien tulosten saatavuus
- staattisten analyysien ja testien vaatimusten ja tulosten saatavuus
- analyysiraportin valmistamiseen kuluva aika
- päällekkäin ja jälkikäteen tehdyt analyysit.

Kelpoistaminen kehitetyllä menetelmällä voidaan tehdä myös, kun suunnittelu ja kehitys ovat täysin valmiita koodausta ja testauksia myöten. Suositeltavampaa kuitenkin

lähinnä prosessien ohjattavuuden kannalta on analysoida mahdollisimman varhaisissa elinkaaren vaiheissa ja sopivissa kohdin vaiheen valmistumisastetta. Jälkikäteen analysoitaessa syötedokumenttien saatavuus ja laatutaso eivät tosin ole niin ongelmallisia kuin päällekkäisessä analysoinneissa, joissa dokumentit saadaan prosessien edistyessä.

Tässä julkaisussa tarkasteltiin vain päällekkäistä analyysivaihtoehtoa, koska se suo nopean ja varhaisen ohjausmahdollisuuden sekä ohjelmistotuotannon prosesseihin että myöhempien vaiheiden analyyseihin. Päällekkäisen analyysin valitseminen vaikuttaa kelpoistamisen suunnitteluun siten, että analyysien aikataulutus pitää sovittaa ohjelmistotuotannon aikataulutukseen. Muutokset jommassakummassa aikataulussa vaikuttavat toiseen, joten suunnitelman tekemisessä pitää ottaa riittävästi huomioon vaihtoehtoisia ratkaisuja analyysitekniikoiden valinnassa sekä väljyyttä kummassakin aikataulutuksessa.

Lähdeluettelo

Beizer, B. 1990. Software Testing Techniques. 2. p. Van Nostrand Reinhold. 503 s. ISBN 0-442-20672-0

Bell, D. et al. 1992. Using Causal Reasoning for Automated Failure Mo-des & Effects Analysis (FMEA). Proceedings of 1992 IEEE Annual Reliability and Maintainability Symposium. S. 343–352.

Bishop, P. (ed.). 1990. Dependability of Critical Computer Systems 3 – Techniques Directory. The European Workshop on Industrial Computer Systems, Technical Committee 7 (EWICS TC7). London: Elsevier Science Publishers LTD.

Bjarland, B. 1986. Ohjelmiston vikasietoisuus. Espoo: Valtion teknillinen tutkimuskeskus. 79 s. + liitt. 17 s. (VTT Tiedotteita 582.)

Boehm, B. W. 1989. Software Risk Management. IEEE Computer Society Press.

Def Stan 00-55: Defence Standards, 1997. Requirements for Safety Related Software in Defence Equipment. Ministry of Defence, Glasgow, UK.

Def Stan 00-56: Defence Standards, 1996. Safety Management Requirements for Defence Systems. Ministry of Defence, Glasgow, UK.

Def Stan 00-58: Defence Standards, 1996. HAZOP Studies on Systems Containing Programmable Electronics. Ministry of Defence, Glasgow, UK.

Deswarete, Y. et al. 1999. SQUALE Dependability Assessment Criteria. Computer Safety, Reliability and Security. In: IFAC SAFECOMP'99. Berlin: Springer-Verlag. S. 27–45. (Lecture Notes in Computer Science, Vol. 1698.)

Friedman, M. A. & Voas, J. M. 1997. Software Assessment: Reliability, Safety, Testability. John Wiley & Sons. 310 s. ISBN 0-471-01009-X

Haikala, I. & Märijärvi, J. 1998. Ohjelmistotuotanto. Suomen Atk-kustannus Oy. 384 s.

Hanna, M. 1993. Maintenance Burden Begging for a Remedy. Datamation. S. 53–63.

Hecht, H. & Hecht, M. 1986. Software reliability in the systems context. IEEE Transactions on Software Engineering, Vol. SE-12, No 1.

IEC 60812, standard. 1985. Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA). International Electrotechnical Commission, Geneve.

IEC 61025, standard. 1990. Fault Tree Analysis (FTA). International Electrotechnical Commission, Geneve.

IEC 61508, standard. 1999. Functional Safety of Programmable Electronic Systems: Generic Aspects. International Electrotechnical Commission. Technical Committee no. 65, Working Groups 9 and 10.

Ippolito, L. & Wallace, D. 1995. A Study on Hazard Analysis in High Integrity Software Standards and Guidelines. National Institute of Standards and Technology. 44 s. (NISTIR 5589.)

ISO/IEC 9126, standard. 1991. Information Technology – Software product evaluation – Quality characteristics and guidelines for their use. International Standard Organisation.

Kletz, T. 1986. HAZOP and HAZAN. Institution of Chemical Engineers, UK.

Laprie, J.-C. (ed.). 1992. Dependability: Basic Concepts and Terminology. IFIP WG 10.4 Dependable Computing and Fault Tolerance. Springer-Verlag, Vienna

Laprie, J.-C. 1985. Dependable computing and fault tolerance: concepts and terminology. Digest of Papers, The Fifteenth Annual International Symposium on Fault-Tolerant Computing. Michigan, USA. S. 2–11.

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

Lawrence, J. D. & Gallager, J. M. 1997. A proposal for performing software safety hazard analysis. Reliability Engineering and System Safety. Elsevier. S. 267–282. (Vol. 55.)

Leveson, N. & Harvey, P. 1983. Analysing Software Safety. IEEE Transactions on Software Engineering. Vol. SE-9, No. 5, s. 569–579.

Leveson, N. & Turner, C. 1992. An Investigation of the Therac-25 Accidents. IEEE Computer, Vol. 26, No. 7, s. 18–41.

Leveson, N. 1986. Software Safety: Why, What and How? ACM Computing Surveys, Vol. 18, No. 2, s. 125–163.

Leveson, N. 1987. What Is Software Safety? COMPASS '87 Computer Assurance. Washington, D.C. S. 74–75.

Leveson, N. 1991. Software Safety in Embedded Computer Systems. Communications of the ACM. Vol. 34, No. 2, s. 34–46.

Leveson, N. 1995. Safeware: System Safety and Computers. A guide to preventing accidents and losses caused by technology. 1. p. New York: Addison-Wesley. 680 s. ISBN 0-201-11972-2

Leveson, N., Cha, C. & Shimeall, T. 1991. Safety Verification of Ada Programs Using Software Fault Trees. IEEE Software, Vol. 8, No 7, s. 48–59.

Lyytikäinen, A. 1987. Käyttövarmuuskäsikirja. Espoo: Valtion teknillinen tutkimuskeskus. 147 s. + liitt. 6 s. (VTT Tiedotteita 678.)

Maskuniitty, M. & Pulkkinen, U. 1995. Fault tree and failure mode and effects analysis of a digital safety function. In: Haapanen, P. (ed.). Advanced control and instrumentation systems in nuclear power plants. Design, verification and validation. Espoo: Valtion teknillinen tutkimuskeskus. S. 158–175. (VTT Symposium 147.) ISBN 951-38-4098-0

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

MIL-STD-882B: Defence standard, 1994. System Safety Program Requirements. Department of Defense, UK.

Montgomery, T. et al. 1996. FMEA Automation for the Complete Design Process. In: Proceedings of the 1996 IEEE Annual Reliability and Maintainability Symposium. S. 30–36.

Musa, J. D. 1999. Software Reliability Engineering in Industry. Computer Safety, Reliability and Security. In: IFAC SAFECOMP'99. Berlin: Springer-Verlag. S. 1–12. (Lecture Notes in Computer Science, Vol. 1698.)

Musa, J. D., Iannino, A. & Okumoto, K. 1987. Software Reliability: Measurement, Prediction: Application. New York: McGraw-Hill Book Company. 621 s. ISBN 0-07-044093-X.

Ormsby, A. et al. 1991. Towards an Automated FMEA Assistant. Teoksessa: Rzevski, G. & Adey, R. (editors). In: Applications of Artificial Intelligence in Engineering VI. New York: Elsevier Applied Science. S. 739–752.

O'Rorke, P. 1992. Failure Modes and Effects Analysis: Opportunities for Automation. Technical Report, University of West Florida.

Peyton, B. & Hess, D. 1985. Software Sneak Analysis. IEEE Seventh Annual Conference of the Engineering in Medicine and Biology Society. The Institute of Electrical and Electronics Engineers.

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

Price, C. et al. 1995. The Flame System: Automating Electrical Failure Mode & Effects Analysis (FMEA). In: Proceedings of 1995 IEEE Annual Reliability and Maintainability Symposium. S. 90–95.

Pulkkinen, U. 1993. Ohjelmoituun automaatioon perustuvien turvallisuustoimintojen luotettavuusanalyysi. Helsinki: Säteilyturvakeskus. 64 s. (STUK-YTO-TR 53.)

Rankin, J. 1973. Sneak Circuit Analysis. Nuclear Safety, Vol. 14, No. 5.

Singpurwalla, N. D. & Wilson, S. 1994. Software reliability modeling. International Statistical Review. S. 289–317.

Storey, N. 1996. Safety-Critical Computer Systems. 1. p. New York: Addison Wessley Longman Inc. 453 s. ISBN 0-201-42787-7

Swanson, E. B. 1976. The Dimensions of Maintenace. Proceedings of the Second International Conference on Software Engineering. IEEE. S. 492–497.

Julkaisija

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

Julkaisun sarja, numero ja raporttikoodi

VTT Tiedotteita 2066
VTT–TIED–2066

Tekijä(t)
Harju, Hannu

Nimeke

Ohjelmiston luotettavuuden kvalitatiivinen arviointi

Tiivistelmä

Tekniset järjestelmät tulevat jatkuvasti yhä enemmän riippuviksi ohjelmistoista. Ohjelmistopohjaiset turvatoimintoja toteuttavat turvallisuuteen liittyvät järjestelmät pohjautuvat tiukkoihin vaatimuksiin ohjelmiston turvallisuuden eheydestä. Niille on olemassa joukko menetelmiä ja tekniikoita, joilla voidaan arvioida ja suunnitella ohjelmiston turvallisuuden eheyttä. Tarve hyödyntää näitä menetelmiä ja tekniikoita on hyvin tiedostettu, mutta korkeitten kustannusten takia niitä ei ole käytetty vähemmän kriittisten ohjelmistojärjestelmien rakentamisessa.

Tässä julkaisussa kuvataan ohjelmiston luotettavuuden kvalitatiivinen kelpoistusmenetelmä, jonka tarkoituksena on arvioida ja ohjata luotettavuusominaisuuksia ohjelmistotuotannon elinkaaren vaiheissa. Ohjelmiston luotettavuusominaisuuksilla tarkoitetaan neljää luotettavuusattribuuttia (toimintavarmuus, käytettävyys, ylläpidettävyys ja turvallisuus), virhemekanismeja (virhe, virhetilanne ja virhetoiminto) sekä keinoja virhemekanismien purkamiseksi tietyn luotettavuuden saavuttamiseksi.

Ohjelmiston luotettavuusvaatimusten kelpoistamismenetelmän kehittämisessä on otetava huomioon muut ohjelmiston virheettömyyteen ja oikeaan suoritukseen tähtäävät menetelmät (luku 4). Menetelmä toteutetaan tietyillä vaara-analyysitekniikoilla (luku 5), jotka sopivilla kriteerivalinnoilla (luku 6) yhdistetään kustannustehokkaaksi menetelmäksi Stérna (luku 7). Stérnaa koekäytettiin yhteistyöyritysten Honeywell-Measurexin ja Neles Automationin ohjelmistoprojekteissa.

Avainsanat
software safety integrity, software reliability, software availability, software maintainability, software fault mechanisms, software dependability requirements validation

Toimintayksikkö
VTT Automaatio, Teollisuusautomaatio, Tekniikantie 12, PL 1301, 02044 VTT

ISBN			Projektinumero
951–38–5766–2 (nid.) 951–38–5767–0 (URL: http://www.inf.vtt.fi/pdf/)			A8SU00291
Julkaisuaika	Kieli	Sivuja	Hinta
Joulukuu 2000	suomi, engl. abstr.	111 s.	C
Projektin nimi		Toimeksiantaja(t) Teknologian kehittämiskeskus (Tekes), Honeywell, Neles Automation, VTT Automaatio	
Avainnimeke ja ISSN VTT Tiedotteita – Meddelanden – Research Notes 1235–0605 (nid.) 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)		Myynti: 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 2066
VTT–TIED–2066

Author(s)

Harju, Hannu

Title

The qualitative assessment of software dependability

Abstract

In recent years, many engineering systems have become increasingly dependent on software. Software based safety related systems which perform safety functions put stringent requirements on the safety integrity of the software involved. Therefore there exists a set of methods and tehcniques that can be used to assess and design safety integrity of software. A need to utilize these methods for less critical items of software is well known, but there isn't reasonably cost-effective methods for software project to use in industrial areas as nuclear power, trafic and medical.

At Chapter seven, a method for software dependability requirements validation to assess and control attributes of software dependability features is described. The dependability features are the four attributes (reliability, availability, maintainability and safety), fault mechanisms (fault, error and failure), and means to remedy fault mechanisms for a level of given dependability.

In developing the method of software dependability requirements validation the other methods aimed to improve correctness of software have to be considered (Chapter 4). The method will be carried on by integrating a set of techniques of specified safety analysis (described at Chapter 5). Techniques will be selected by appropriate criterias (Chapter 6) for conducting cost-effective dependability validation of the software called Stérna. Stérna was tested at software case projects of co-operation company of Honeywell-Measurex and Neles Automation.

Keywords

software safety integrity, software reliability, software availability, software maintainability, software fault mechanisms, software dependability requirements validation

Activity unit

VTT Automation, Industrial Automation, Tekniikantie 12, P.O.Box 1301, FIN–02044 VTT, Finland

ISBN	Project number
951–38–5766–2 (soft back ed.) 951–38–5767–0 (URL: http://www.inf.vtt.fi/pdf/)	A8SU00291

Date	Language	Pages	Price
December 2000	Finnish, Engl. abstr.	111 p.	C

Name of project	Commissioned by
	The National Technology Agency (Tekes), Honeywell, Neles Automation, VTT Automation

Series title and ISSN	Sold by
VTT Tiedotteita – Meddelanden – Research Notes 1235–0605 (soft back ed.) 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)	VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374