Ohjelmiston luotettavuusarvioinnit

Hannu Harju,
VTT Automaatio

Laprien (1985) mukaan luotettavuus on se järjestelmän ominaisuus, jolla voidaan perustella luottamusta järjestelmän tuottamiin palveluihin eli turvautua järjestelmään. Usein luotettavuutta pidetään yleisterminä, jolla tarkoitetaan järjestelmän kaikkia luotettavuusattribuutteja: toimintavarmuutta, käyttövarmuutta, ylläpitovarmuutta ja turvallisuutta, englanniksi reliability, availability, maintainability and safety, joista lyhennys: RAMS.

Oikea RAMS-analyysimenetelmien hyödyntäminen edellyttää niiden tarvetilanteen tarkkaa määrittelyä. Tekniikkojen käyttö pitää liittää saumattomasti muihin projektitoiminnassa tai lopputuotteen tarkasteluissa oleviin menetelmiin: ohjelmiston staattisiin ja dynaamiisiin analyyseihin, testeihin, katselmuksiin ja tarkastuksiin. Tulee tuntea kohteen vikamekanismit ja keinot vioista suojautumiseen.

Ohjelmoinnin RAMStekniikkaa

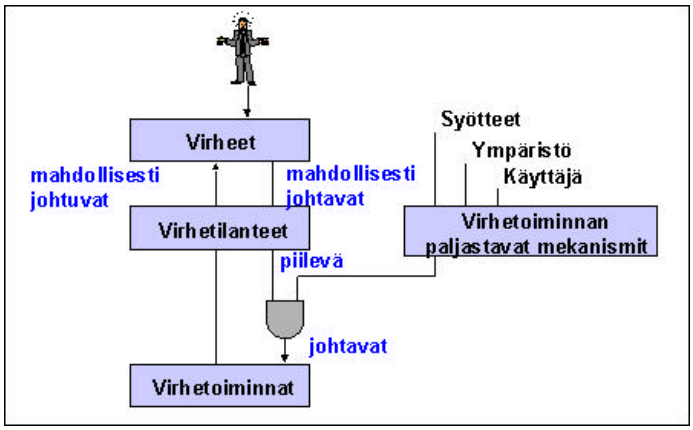
Luotettavuustekniikkaan liittyvät seuraavat kolme käsitettä:

1. virheet
2. suojautumiskeinot
3. RAMS-attribuutit

Ohjelmiston luotettavuuden heikennyksiä kuvaavat virhekäsitteet ohjelmistovirhe (fault), virhetilanne (error) ja virhetoiminta (failure) (Bjarland 1986).

Virhetoiminta on ohjelman ulkoisesti havaittavan palvelun poikkeama ohjelman spesifikaation mukaisesta toiminnasta eli siitä mitä järjestelmään pitäisi tehdä. Virhetilanne on se osa järjestelmän tilaa, joka voi mahdollisesti johtaa sopivien syötteiden ja ympäristön vallitessa virhetoimintaan (kuva 1). Virhe on virhetilanteen todellinen (havaittu) tai oletettava syy.

Suojautumiset luokitellaan usein vian ennaltaehkäisyyn (fault prevention), vikasietoisuuteen (fault tolerance), viasta toipumiseen (fault removal) ja vian ennustamiseen (fault forecasting). Suojautumismenetelmät ovat tekniikoita, ratkaisuja ja työkaluja, joita tarvitaan luotettavan tuotteen suunnittelussa ja tuottamisessa.



Kuva 1. Ohjelmiston virhekäsitteet: virhe, virhetilanne ja virhetoiminta. Virhe johtaa virhetilanteen kautta virhetoimintaan tietyissä olosuhteissa.

Virheiden syntyä lopputuotteeseen voidaan ennaltaehkäistä joko välttämällä niiden syntyä jo alunperin tai tunnistamalla ja eliminoimalle ne. Vikasietoisuudella tarkoitetaan järjestelmän kykyä toimia virheettömästi vikojen yhteydessä.

Toimintavarmuus on järjestelmän kyky toimia seuraavilla ehdoilla:

- järjestelmä toimii sille spesifioidulla tavalla
- aikajako on määrätty
- käyttäytyminen on loppukäyttäjän hyväksymä
- ympäristö on käyttöön tarkoitettu.

Toimintavarmuus liittyy järjestelmän ennakoitavaan virhetoimintaan, käyttövarmuus taas käyttövalmiuteen. Käyttövarmuutta tarvitsevat kaikki järjestelmät jossakin määrin, toimintavarmuutta, ylläpidettävyyttä ja turvallisuutta vain tietyt järjestelmät. Ylläpidettävyys on korjausten ja uudelleenkäytön tai -kehityksen suorittamisen tarkoituksenmukaisuutta. Toisin sanoen ylläpidettävyys on järjestelmän kyky

- kelpuuttaa tehdyt muutokset ja palata käyttötilaan määrätyssä ajassa virhetoiminnan jälkeen
- päivittää ja parantaa järjestelmää
- mukauttaa sitä muuttuvaan ympäristöön.

Ylläpidettävyys on merkittävä järjestelmälle myös siksi, että sitä tarkastellaan kaikissa järjestelmän kehityksen vaiheissa. Virheet ylläpidossa näkyvät kaikissa muissa RAMS-attribuuteissa.

Toimintavarmuudelle läheinen käsite on turvallisuus, joka merkitsee sitä, että tietyillä ehdoilla järjestelmä ei voi joutua kriittiseen tilanteeseen tai tietyssä ympäristössä onnettomuuteen.

Luotettavuusevidenssien kokoaminen

Vakuuttuminen ohjelmiston riittävästä luotettavuudesta voi perustua erilaisten evidenssien tuottamiseen ja yhdistämiseen sopivalla tavalla (kuva 2). Lupaavimmaksi yhdistämiskeinoksi on osoittautumassa Bayesian Belief Network (Littlewood et al. 1996). Kolmenlaiset argumentit tuottavat evidenssejä: analyyttiset, eksperimentaaliset ja konformitiiviset (Harju et al. 1998).

Analyyttiset argumentit pureutuvat muita argumentteja paremmin ohjelmistotuotannon elinkaaren alkuun, esisuunnitteluun ja määrittelyvaiheisiin. Analyyttisesti voidaan luotettavuuden arvioinnin lisäksi myös ohjata suunnittelua mm. kehittämällä testausta ja laadunvarmistusta. Eksperimentaalisista argumenteista tärkeimpiä ovat testit, jotka ovat myös tärkein luotettavuuden osoittamiskeino ja ne soveltuvat parhaiten valmistumassa olevalle tuotteelle eli elinkaaren loppuun. Siinä missä analyyttiset tarkastelevat mahdollisuuksia, eksperimentaaliset mahdollisuuksien toteu-

tumista. Konformitiivisin keinoin pyritään osoittamaan deterministisen vaatimusten, standardien, suunnittelusääntöjen ja laatujärjestelmien toteutumista.

Evidenssien tuottajista käsitellään tässä esityksessä analyyttisia ja niistä edelleen luotettavuudelle ominaisinta eli RAMS-analyysitekniikoita. Niitä käytetään yleensä silloin kun kyse on joko hyvin kriittisistä systeemeistä tai halutaan kehittää järjestelmän tuotantoa.

Ohjelmiston RAMS-tekniikat

RAMS-analyysitekniikoilla saavutettavat hyödyt: RAMSien kvalitatiiviset ja kvantitatiiviset arviointitulokset

- seurausten arviointi
- kriittisyyden arviointi
- virheiden tunnistus
- laadunvarmistuksen, analysointien, testien kohdistaminen ja tukeminen
- ohjelmistotuotannon kehittäminen.

Seurausten jäljittämiseen tarkoitettuja analyyseja kutsutaan induktiivisiksi analyyseiksi ja niistä tärkeimpiä ohjelmiston kannalta on ohjelmiston vika-, vaikutus- ja kriittisyysanalyysi (Software Failure Modes, Effect and Criticality Analysis, SFMECA). Toiseen suuntaan eli syiden tunnistamiseen kohdistuvia analyyseja kutsutaan deduktiivisiksi analyyseiksi ja niistä tärkein on ohjelmiston vikapuuanalyysi (Software Fault Tree Analysis, SFTA).

Induktiiviset päättelyt

RAMS-analyysien suoritusperiaatteena on käsitellä systeemitaso ja varhaiset elinkaaren vaiheet niin tarkasti että ohjelmiston kooditarkastus jää pieneksi. Systeemitason, projektoinnin, määrittely- ja suunnitteluvaiheiden tarkastelun kautta tiedetään mitä ongelmakohtia ohjelmakoodista haetaan.

Ylimmän tason analyysit aloitetaan tunnistamalla kriittiset funktiot jonka jälkeen jatketaan yksityiskohtaisella induktiivisella menetelmällä joko SFMECA:lla, poikkeamatarkastelulla (HAZOP) tai ohjemiston poikkematarkastelulla (Software Deviation Analysis) tai tapahtumapuuanalyysilla.

SFMECA on kvalitatiivinen tekniikka, jolla selvitetään kohteen eri osien vioittumistavat sekä määritellään niiden vaikutukset ja kriittisyydet kohteen muihin osiin sekä kohteelta vaadittuun toimintaan. SFMECA 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 dominoivia yksittäisvikoja ja paljastetaan piileviä vikatapahtumia.
SFMECA:ta sovellettaessa myös ohjelmistokoodiin analysoijat listaavat kohteen komponenttien mahdolliset viat ja tunnistavat potentiaaliset seuraukset systeemitasolle.

SFMECA soveltuu kaikkiin vaiheisiin vaatimusmäärittelystä koodausvaiheeseen, tosin suoraan koodiin kohdistuvista analyyseista ei ole erityisemmin referoituja lähteitä. Eräs syy tähän on menetelmän systemaattisessa luonteessa, joka helposti johtaa resursseja vaativaan detaljisuuteen. VTT Automaatio oli tarkastelemassa ESA:n (European Space Agency) erään säsatelliitin laajaa ja kriittistä ohjelmistoa mm. kyseisellä menetelmällä. SFMECA:ta käytettiin funktionaalisella tavalla tunnistamaan kriittisiä ohjelmistoja vaatimusspesifikaatiotasolla. Tunnistettiin funktioiden poikkeamien kriittiset seuraukset, määriteltiin suojaavia toimenpiteitä, testiohjeita ja laitevikojen vaikutuksia ohjelmistoon (HSIA: Hardware Software Interaction Analysis).

Kuva 2. Ohjelmiston luotettavuusarviointi vaatii eri menetelmillä koottuja todisteita.

Tapahtumapuuanalyysilla tutkitaan voiko inhimillinen virhe tai laitteen toimintahäiriö aiheuttaa hyväksymättömän vikaantumisen. Analyysi aloitetaan varsinaisesta vioittumisen syystä eli alkutapahtumasta ja jatketaan tunnistamalla viasta aiheutuvia tapahtumia. Alkutapahtumasta piirretään kaksi viivaa: ensimmäinen positiiviselle seuraukselle ja toinen negatiiviselle. Tarkastelua jatketaan sitten kustakin seuraustapahtumista siihen asti kun kaikki merkittävät seuraukset on käsitelty.

Deduktiiviset päättelyt

Ohjelmiston vikapuuanalyysi (Software Fault Tree Analysis, SFTA) on looginen kaavio, joka esittää kohteen kriittisen vikautumisen (top-tapahtuma) 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 korjaavia toimenpiteitä joilla tapahtumaketjut voidaan pysäyttää. Viat ja virheet voivat aiheutua ohjelmoinnista, suunnittelusta, inhimillisistä tekijöistä ja/tai laitteistosta.

Leveson (1983, 1991) on kuvannut ohjelmiston vikapuutekniikkaa. Periaatteena niissä on kuvata ohjelmistokäskyt minivikapuina kuvan 3 mukaisesti. Minivikapuut yhdistetään systeemitason vikapuihin tai kriittisiin tapahtumiin sekä jatketaan virheiden ja virhetilanteiden etsintää ohjelman lähdekoodin suoritusjärjestystä noudattaen. Laitteiston ja ohjelmiston vikapuiden yhdistäminen on merkittävää siksi, että monet kriittiset seuraukset voivat aiheutua ohjelmistovirheen, laitteistovian tai inhimillisen virheen kombinaatiosta.

SFTA soveltuu kaikkiin muihinkin ohjelmistoprosessin vaiheisiin kuin koodaukseen kunhan vain top-tapahtumat ovat tiedossa ennen analyysiin ryhtymistä.

Edellä mainitussa ESA:n sääsatelliitin ohjelmiston luotettavuusanalyysissä vikapuun top-tapahtumat haettiin SFMECA:lla. Top-tapahtumista lähdettiin jäljittämään kriittisiä ohjelmistokohtia SFTA:lla, jota sovellettiin HOOD-suunnittelun (Hierarcical Object Oriented Design) kautta ADA -koodiin. Analyysien tulosten pohjalta tiedettiin tarkasti mitä koodista haluttiin tarkastaa.

SFTA:n eräs merkittävä hyöty on sen kyvyssä laskea top-tapahtuman todennäköisyys. Kuitenkin, ilman Bayesian Belief Networkin kaltaista menetelmää ohjelmistovirheiden ja muiden vastaavien systemaattisten virheiden (mm. inhimilliset virheet) mukaanotto kvantitatiiviseen tarkasteluun on hankalaa. Muita merkittäviä etuja ja haittoja ovat seuraavat:

- Paljastaa moninkertaisia eri järjestelmän osissa syntyneitä vikoja
- Voidaan hyödyntää myöhemmin vikapuuhun taltioitunutta informaatiota mm. suunnitteluun ja testaukseen.
- Voidaan tarkastella koko järjestelmää (ohjelmisto, laitteisto ja käyttöliittymä).
- Ei ole käytännöllinen järjestelmille joissa on lukuisia kriittisiä vikoja.
- Ajastustekijät vaikeita liitettäviä vikapuihin.

Vikapuu kuvaa tapahtumat loogisesti. Siten logiikkaa noudattavat tapahtumasarjat voidaan kuvata. SFMECA:lla usein etsitään kriittiset tapahtumat, joita sitten tarkemmin tutkitaan SFTA:lla. Myös päinvastaista menettelyä on käytetty (Maskuniitty et al. 1995), tällöin FTA:lla on ensin tunnistettu lyhyet minimikatkosjoukot joita sitten on tarkemmin tutkittu FMECA:lla.

Muita RAMS-analyyseja

Oikopolkuanalyysin tavoitteena on havaita odottamaton polku tai logiikkavuo, joka tiettyjen ehtojen vallitessa tulee suoritetuksi ja joka voi johtaa ei-toivottuun toimintaan tai estää halutun toiminnon. Polku voi muodostua laitteistosta, ohjelmistosta, operaattorin toimenpiteistä tai näiden yhdistelmistä. Oikopolut eivät ole laitteiston virhetoiminnon tuloksia vaan järjestelmään tai koodiin tarkoituksettomasti sijoitettuja piileviä ehtoja, jotka voivat aiheuttaa virhetoimintoja ehtojen tullessa voimaan.

Alustava turvallisuusanalyysi, Preliminary Hazard Analysis, PHA on ylimmän tason kvalitatiivinen menetelmä ohjelmiston kehityksen elinkaaren varhaisissa vaiheissa. Sillä tunnistetaan kriittisiä vaaratekijöitä ja määritellään järjestelmän RAMS-vaatimukset. Tarkastelu perustuu erilaisiin läpikäyntityylisiin tarkasteluihin ja tarkistuslistoihin. Tuloksena alustavasti luokitellaan järjestelmän ohjelmistosta /laitteistosta aiheutuvat riskit ja päätetään lisätarkastelujen kohdistamisesta, mm. kohdistamalla RAMS-ominaisuuksia lisääviä järjestelyjä mm. suunnittelulla.

Ohjelmistovaatimusten turvallisuusanalyysi perustuu PHA:n ja mahdollisen esisuunnittelun tuloksiin ja menetelmä onkin PHA:n jatkoa tunnistettaessa kriittisiä ohjelmiston spesifikaatiovirheitä. Kriittisyyden määrää riski, joka liittyy spesifikaatiovirheisiin.

Petriverkot ovat ensisijassa suunnittelumenetelmä, jonka käyttö turvallisuusanalyyseihin edellyttäisi ohjelmiston suunnittelua petriverkkomenetelmällä. Menetelmän käyttö vaatii eksperttejä, on aikaavievä ja kallis. Soveltuu ohjelmiston vaatimusten spesifioinnista koodaukseen.

Yhteisvika-analyysi on tärkeä erityisesti jos järjestelmän arkkitehtuuri voi koostua redundanttisista osista, yhteisistä tai samantyyppisistä komponenteista tai samasta ohjelmistosta. Kriittisen yhteisvian, eli yksittäisvian vaikutuksesta aiheutuvan usean komponentin kriittisen vikaantumisen mahdollisuus kasvaa näissä tapauksissa. Vikoja, jotka vahingoittavat varmistuksia tai riippumattomia olettamuksia järjestelmän toiminnasta kutsutaan yhteisvioiksi. Yhteisvika-analyysilla voidaan tarkastella kohteen yhteisvikoja eri tavoin riippuen yhteisvian vaikutusalueesta.

Markovin mallinnus on klassinen tekniikka, jolla selvitetään dynaamisten järjestelmien aikariippuvaa käyttäytymistä. Markovin ketjussa siirtymät tilojen välillä tapahtuvat vain diskreeteillä ajanhetkillä. Markovin mallinnus on

ıva 3. Ohjelmakäskyn IF-THEN-ELSE vikapuu (Leveson 91).

suosittu ohjelmistopohjaisten järjestelmien kvantitatiivinen laskentamenetelmä.

VIITTEET

Bjarland B. (1986).
Ohjelmiston vikasietoisuus. VTT Tiedotteita 586, Espoo 1986: 97 s..

Harju H, Haapanen P, Karvonen I. and O. Ventä (1998).
Software dependability assessment. Automation Technology Review 1998, VTT Automation, Espoo 1998: 34-40.

Laprie J.-C. (ed.) (1998).
Dependability handbook. Laboratory for Dependability Engineering, LAAS Report no 98-346, Aug. 1998: 291p.

Leveson N.G. and M.P.E. Harvway (1983).
Analysing Software Safety. IEEE Trans. Software Eng. Sep.1983: 569-579.

Leveson N.G, Cha S.S and T.J. Shimeall (1991).
Safety Verification of ADA Programming Using Software Fault Trees. IEEE Software, Jul.1991: 49-59.

Littlewood M.N.B. and N.Fenton (1996),
Applying Bayesian Belief Networks to System Dependability Assessment. Proceedings of Safety Critical Systems Club Symposium, Leeds, 6-8, February 1996. Published by Springer-Verlag.

Maskuniitty M. and U. Pulkkinen (1995).
Fault tree and failure mode and effects analysis of a digital safety functions. VTT Symposium 147, VTT, Espoo 1995: 158-175.

Hannu Harju,
VTT Automaatio