Ohjelmistovirheet avaruustutkimuslaitteistoissa

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Toukokuu 2024
Linda Kallio-Kujala

TURUN YLIOPISTO
Tietotekniikan laitos

Linda Kallio-Kujala: Ohjelmistovirheet avaruustutkimuslaitteistoissa

LuK-tutkielma, 20 s.
Tietojenkäsittelytiede
Toukokuu 2024

Tämä kandidaatintutkielma on kirjallisuuskatsaus siihen, millaisia ohjelmistovirheitä avaruuslaitteistojen tietokoneiden ohjelmissa on esiintynyt, miten niistä aiheutuvat ongelmat ratkaistaan ja mikä vaikeuttaa ohjelmistokehitystä luotaimiin ja raketteihin. Tärkeimpinä lähteinä toimivat Grottke ym. tutkimus "Epirical Investigation of Fault Types in Space Mission System Software" ja Trivedi ym. tutkimus "Epirical Investigation of Fault Repairs and Mitigations in Space Mission System Software". Nämä tutkimukset perustuvat NASAn/JPLn 18 lennolta saatuihin virheraportteihin ja niistä eriteltyihin uniikkeihin ohjelmistoperäisiin poikkeamiin.
Tutkimuksissa ohjelmistovirheet jaetaan kahteen eri luokkaan: Bohrbugeihin ja Mandelbugeihin. Bohrbugin aktivoi yksinkertaiset olosuhteet ja se on helposti havaittavissa, Mandelbugi taas aktvoituu monimutkaisten tapahtumasarjojen seurauksena tai epäsuorasti. Bohrbugeille suositeltu käsittelytapa on korjaaminen. Mandelbugeille suositellaan korjaamisen lisäksi uudelleen yrittämistä, uudelleenkäynnistystä tai muuta suoritusympäristöä muuttavaa tapaa.
Tutkituilla lennoilla yleisimpiä olivat katastrofaalisia seurauksia aiheuttavat Bohrbugit ja toiseksi yleisimpiä Mandelbugit, joiden seuraukset olivat mitättömiä. Virhetyypillä ei ollut yhteyttä siihen, oliko virheen tapahtumisen todennäköisyys hyväksyttävällä vai ei hyväksyttävällä tasolla.
Virheiden vähentämiseksi JPL on kehittänyt niin ohjelmointistandardeja kuin myös ohjelmoijien sertifiointikurssin. JPL vaatii kaikki avaruuslaitteistojen koodin parissa työskentelevät ohjelmistokehittäjänsä läpäisemään sertifiointiprosessin, johon kuuluu kolme eri moduulia tärkeimmistä aiheista sekä koe joka moduulin jälkeen. Lisäksi JPL on luonut standardin käytettävälle koodille, joka perustuun aiemmilla lennoilla tapahtuneisiin vaaratilanteisiin. Koontiversiot tarkastetaan staattisella analyysillä ja eriävyydet korjataan.

Asiasanat: ohjelmistovirheet, ohjelmistot, luotaimet, avaruusraketit

Sisällys

1 Johdanto

Avaruuslennolla kaiken pitää toimia ei pelkästään itse raketin tai luotaimen sisällä olevissa tietokoneissa, vaan myös maanpinnalla niihin yhteydessä olevissa tietokoneissa. Näiden suunnittelussa ongelmia voivat tuottaa niin suorat ohjelmointivirheet, kuin puhtaat väärinkäsitykset siinä, miten ohjelmiston tulisi toimia [1]. Lisäksi ohjelmisto- ja laitteistokehittäjien täytyy ottaa huomioon avaruuden ja paino- ja tilarajoitteiden tuomat ongelmat. Tämä tekee heidän työstään huomattavasti monimutkaisempaa Maassa käytettävien tietokoneiden suunnitteluun verrattuna [2].

Avaruusjärjestöt ovat olleet mukana tietokoneiden ja ohjelmistojen kehityksessä aina 1960-luvulta lähtien. Ne eivät kuitenkaan suoraan pyrkineet vauhdittamaan tietokoneiden kehitystä. Kehitys tapahtui avaruusohjelman tarpeen tai ongelman takia ja ratkaisusta syntynyt teknologia oli usein hyödyllistä myös kaupallisesti tuotetuissa tietokoneissa. Siinä missä aluksissa olevat tietokoneet vauhdittivat tekniikan kehitystä, oli maassa olevilla tietokoneilla isompi vaikutus käyttöjärjestelmien ja ohjelmien kehitykseen. Niiden tarvitsisi esimerkiksi käsitellä suuria määriä dataa, mikä oli tarpeellista myös kaupallisilla tietokoneella. Miehittämättömiin luotaimiin kehitettiin tietokoneet lähes alusta alkaen uudelleen niiden uniikeista vaatimuksista johtuen. Miehitettyjen lentojen tietokoneet kehitettiin edellisten lentojen koneiden pohjalta, maa-asemien tietokoneet puolestaan kaupallisia tietokoneita pohjana käyttäen. [3]

Kun tietokoneita alkoi ilmestyä avaruusaluksiin ja -laitteistoihin oli ohjelmistokehitys vielä hyvinkin luovaa, eikä alalla ollut vielä paljonkaan kehitettyjä periaat-

teita. Tarvittiin varsinkin tapoja kiertää kustannusten ja tila- ja painorajoitteiden tuomat ongelmat. Kehittäjät huomasivat pian että oli kätevämpää kirjoittaa koodi moduuleina, jolloin sen osia voitiin käyttää kätevämmin myöhemmissä ohjelmissa ja osa koodista voitiin helpommin tallentaa eri sijainteihin. Avaruusaluksia haluttiin koko ajan monipuolistaa tehtävien mutkistuessa. Varsinkin miehitettyjä kuulentoja suunnitellessa tarve aluksen sisäiselle tietokoneelle tuli selväksi. Tehtävä vaati navigointia, ohjausta ja lentämistä kontrolloivaa konetta, sillä haluttiin välttyä signaalien ylisaturoitumiselta ja mahdolliselta vihamieliseltä häirinnältä. Tämän lisäksi signaalilta meni yli sekunti kulkea Kuun ja Maan välinen matka, mikä oli suuri riski vaarallisessa laskeutumisessa Kuun pinnalle. [3]

Suuret välimatkat, Maan signaalien ulkopuolelle tapahtuvat lennot sekä avaruuden armottomuus vaativat laitteistoiden koneilta sekä luotettavuutta että virheensitokykyä. Suuret määrät koodia ja monimutkaiset laitteistot ovat omiaan aiheuttamaan erilaisia ohjelmistovirheitä. Vaikka avaruusjärjestöissä pyritäänkin palkkaamaan kokeneita ohjelmistokehittäjiä, ovat erilaiset bugit ja virheet silti päänvaivana. Yksinkertaisetkin virheet ja huonosti dokumentoitu koodi voivat aiheuttaa avaruuslentojen tapauksessa katastrofaalisia virheitä ja miljoonien tappiot niin avaruusjärjestölle kuin luotainten ja mittauslaitteiden omistajille. Tämän vuoksi on tärkeää, että avaruudessa käytettävät ohjelmistot suunnitellaan sietämään ja palautumaan virhetilanteista mahdollisimman hyvin. [2]

Tässä tutkielmassa pyritään vastaamaan seuraaviin tutkimuskysymyksiin:

1. Mikä hankaloittaa ohjelmistojen suunnittelua avaruudessa toimiviin laitteistoihin?
2. Millaisia virheitä niissä esiintyy?
3. Miten virheitä tulisi käsitellä?

Ratkaisut pohjautuvat IEEEXploren ja Google Scholarin avulla löydettyihin tieteellisiin tutkimuksiin aiheesta ja raportteihin tapahtuneista ohjelmistovirheistä. Tutkielmassa pääpaino on NASAn lennoilta saadussa datassa ja tuloksissa. Hakusanoina käytettiin seuraavia: spacecraft software, space software, software fault, bugs in spacecraft software.

Luvussa 2 esitellään eri virhetyypit ja niiden esiintyvyystiheys NASAn avaruuslennoilla. Lisäksi perehdytään siihen, millaisia uniikkeja ongelmia ja rajoitteita avaruus ja raketit ympäristönä tuovat ohjelmistokehitykseen. Luvuissa 3 kerrotaan miten erilaisia bugeja päädytään yleensä ratkaisemaan. Luvussa 4 esitellään pari tunnetuimpaa ohjelmistovirheestä johtunutta luotaimen ja raketin menetystä ja katsotaan, miten aiemmin esiteltyjä mitigointitapoja oltaisiin voitu käyttää näissä tapauksissa laitteiston pelastamiseksi.

2 Ohjelmistovirhetyypit, niiden esiintyvyys ja muut ongelmat

Kaikissa, paitsi kaikista yksinkertaisimmissa ohjelmistoissa on virheitä, eli bugeja. Erityyppisille virheille, joita ohjelmiston ajon ja testauksen aikana voi tapahtua, ei ole virallisia termejä tai termejä käytetään epäjohdonmukaisesti alan kirjallisuudessa eikä niitä kummemmin selitetä. Tässä tutkielmassa käytetään Grottke ynnä muiden (2010) määritelmiä ja nimiä erilaisille bugeille. [1]

2.1 Ohjelmistovirhetyypit

Bugit luokitellaan kahteen päätyyppiin, Bohr- ja Mandelbugeihin. Bohrbugi on helposti havaittavissa oleva virhe, joka toistuu tietyssä ympäristössä ja/tai tapahtumasarjassa, sillä sen tapahtuminen vaatii yksinkertaiset olosuhteet verrattuna muihin virhetyyppeihin. Toinen päätyyppi ovat Mandelbugit ja ne ilmestyvät monimutkaisen tapahtumasarjan seurauksena tai epäsuorasti. Monimutkaisuutta voivat aiheuttaa viivytys bugin syntymisen ja esiintymisen välillä tai epäsuorien osatekijöiden vaikutus. Osatekijöitä voivat olla esimerkiksi laitteisto, käyttöjärjestelmä, muut sovellukset, syötön ajoitus, kalenteriaika ja operaatioiden ketjutus, varsinkin jos operaatiot olisi voinut ketjuttaa jossain muussa järjestyksessä, joka ei olisi aiheuttanut virhettä. Mandelbugien syy on vaikeaa löytää tai sen aiheuttamat virheet eivät tapahdu joka kerta vaikka ympäristö pysyisi samana. Jokainen virhe on joko Bohrbugi

tai Mandelbugi. Näiden kahden lisäksi mainittakoon vielä käynnissäoloaikaan liittyvät AR-bugit (eng. aging-related bugs), jotka aiheuttavat sitä enemmän virheitä mitä kauemmin niiden annetaan toimia tai huonontaa suoritusta. Ne ovat Mandelbugien alatyyppi. [1]

Nämä määritelmät eivät perustu siihen, miten virhe toimii vain tietyssä tapauksessa, vaan paremminkin niihin potentiaalisiin ominaisuuksiin, joita sillä ilmetessään voi olla. Virhe luokitellaan Mandelbugiksi jos se on potentiaalia aiheuttaa virheitä, joita ei pystytä systemaattisesti toistamaan, ja AR-bugiksi, jos sillä on potentiaali tuottaa aina vain lisääntyviä virheitä. Luokittelu perustuu siis bugien luontaisiin ominaisuuksiin paremminkin kuin siihen, millaisia virheitä ne ovat jo tuottaneet. Mandelbugien kompleksisuuden takia on oletus, että suurin osa ohjelmistoon testauksen jälkeen jääneistä virheistä on tähän luokkaan kuuluvia. Aiheesta tehdyt tutkimukset eivät kuitenkaan ole tarpeeksi kattavia tähän lopputulemaan tulemiseen. Se, kuinka suuri osa bugeista kuuluu mihinkin luokkaan auttaa bugien huomaamisessa, niiden vaikutusten naamioinnissa ja kaikista todennäköisimmin vaikeasti löydettäviä bugeja sisältävien komponenttien tunnistamisessa. [1]

2.2 Virheiden esiintyvyys

Jet Propulsion Laboratory (JPL) on tuottanut tutkimuksia eri avaruustutkimuslaitteistoissa tapahtuneista poikkeamista osana heidän Ultrareliability ohjelmaansa. Näitä poikkeamia ovat muun muassa käyttäjän väärinymmärrykset sekä laitteisto- ja ohjelmistoviat. Tutkimus valaisi eri kategorioihin kuuluvien poikkeamien suhteellisia määriä (relative proportions), mihin aikaan poikkeamia sattui tehtävien aikana ja miten ne yleensä korjattiin. Tutkimustulokset kertoivat ohjelmistoperäisten poikkeamien olevan suuressa osassa. Galileo-luotainta käsitelleessä tutkimuksessa ohjelmistovirheet olivat vähän alle puolesta aina kahteen kolmasosaa asti syynä havaituille poikkeamille. Tutkimuksessa ei kuitenkaan eritelty eri virhetyyppejä. [2]

Grottken ynnä muiden (2010) tutkimuksessa tarkasteltiin 18 mennyttä tai parhaillaan menossa olevaa NASAn/JPLn avaruuslentoa. Niillä raportoitiin yhteensä yli 13 000 poikkeamaa, joista 653 oli ohjelmistoperäisiä. Näistä 76 :n pääteltiin kuvauksen mukaan olleen käyttäjästä lähtöisin olevia, eikä siis ohjelmistovirheiksi laskettavia, ja 57 oli jo aiemminkin raportoituja ohjelmistovirheitä. Jättämällä nämä 133 poikkeamaan huomiotta tarkastelemme siis 520 jäljelle jäävää lennon aikana tapahtunutta, ohjelmistoperäistä ja uniikkia ongelmaa. 319 näistä virheistä luokiteltiin Bohrbugeiksi, 176 Mandelbugeiksi ja 23 AR-bugeiksi. 11 virhetyyppiä ei pystytty määrittelemään annettujen tietojen perusteella. [1]

Tutkimuksessa huomattiin, ettei virheen tyypillä ja kriittisyydellä ole yhteyttä. Kriittisyydellä tarkoitetaan tässä sitä, kuinka vakavan riskin bugi aiheuttaa. Tutkimuksessa virheiden riskit luokiteltiin neljään eri vakavuusluokkaan: ei hyväksyttävä riski, hyväksyttävä riski, ei huomattavaa riskiä tai ei riskiä ollenkaan. Sekä Bohrettä Mandelbugit voivat aiheuttaa yhtä kriittisiä ongelmia. Virhetyypin ja sen aiheuttamien seurausten välillä taas olisi mahdollisesti yhteys. Bohrbugit olivat yliedustettuina katastrofaalisissa seurauksissa, kun taas Mandelbugien seuraukset olivat vastaavasti useimmin olemattomia. [1]

Lisäksi tutkittiin bugien määrän yhteyttä lennon kestoon ja laukaisuajankohtaan. Tähän valittiin vain kahdeksan lentoa, sillä kymmenellä muulla lennolla virheiden määrät olivat huomattavan vähäisiä, vain noin 3,9 virhettä per lento. Näillä kahdeksalla lennolla poikkeamia taasen oli yhteensä yli 400. Eniten Bohrbugeja löytyi kolmelta neljästä uusimmasta lennosta. Tämä saattoi kirjoittajien mukaan johtua ehkä erilaisesta kehitysprosessista, paremmasta Mandelbugien etsimisteknologiasta tai siitä, että ohjelmien suoritusympäristö on paremmin kontrolloitu. Lisäksi käyttöön on voitu ottaa ohjelmistoreplikaatioita ja ohjelmistoa nuorentavia tekniikoita, jolloin koodissa olevat Mandelbugit eivät joko aktivoidu ollenkaan tai esiinny suorituksen aikana. Vanhempien ja uudempien lentojen välillä ei kuitenkaan esiintynyt

suuria eroja käytetyissä virhetoleranssistrategioissa, joten erilaiset strategiat tuskin olivat syynä eroihin. [1]

2.3 Ulkoiset haasteet

On arvioitu, että Kansainvälisen Avaruusaseman tietokoneilla on yli puolitoista miljoonaa riviä koodia. Kun Expedition 10 teki matkan asemalle vuonna 2005, oli yksi heidän tehtävistään ohjelmistopäivitysten asennus. Näillä päivityksillä päästiin eroon siihen asti käytössä olleista 300:ta eri väliaikaisesta korjauksesta koodissa. Avaruus tuottaakin monia uusia ongelmia ohjelmistokehittäjille. [2]

Kun suunnitellaan avaruudessa käytettävää ohjelmistoa, on otettava huomioon se, ettei sitä välttämättä käytetä kuin vasta kuukausien, joskus jopa vuosien päästä laitteiston laukaisusta avaruuteen. Lisävaativuuta tuo mahdollinen uudelleenohjelmointi lennon aikana, mitä pidetään laajasti sekä vaikeana että virhealttiina. Ohjelmointitiimit ovat kehitelleet erilaisia strategioita vähentääkseen mahdollisten virheiden määrää näissä tilanteissa. Niitä ovat esimerkiksi vaihtoehtoisten käskysarjojen luonti ja hybridimalli, jossa uutta koodia voidaan ladata vain monien tarkistuksien jälkeen ja vain tietyissä vaiheissa lentoa. Muissa tapauksissa voidaan käyttää vain valmiita komentosarjoja. [2]

Vikatilanteet, fyysisen laitteiston hajoaminen tai aluksen positio voivat myös estää yhteyden maahan. Planeettojen ja Auringon läpi ei pystytä lähettämään signaaleja ilman erillistä linkkinä toimivaa luotainta. Ongelmia tuottavat myös muut tähdet. Supernovista peräisin oleva galaktinen kosminen säteily on otettava huomioon ohjelmistojen suunnittelussa, sillä avaruudessa toimivat tietokoneet vaativat säteilysuojattuja prosessoreja. Kaupalliset prosessorit eivät tätä säteilyä kestä. Ohjelmoijien täytyykin tietää prosessorien uniikit ominaisuudet ja lisäksi toimia ilman suurta määrää kaupallisille prosessoreille tehtyjä ohjelmistonkehityssovelluksia. [2]

Laitteistojen suunnitteluun ja rakentamiseen tarvitaan monitieteellinen tiimi,

mikä itsessään lisää mahdollisuutta väärinkäsityksille alojen erilaisten termistöjen ja käytäntöjen takia. Erikoislaitteistosta seuraa myös se, että jotkut laitteistot saattavat vaatia osia, joita on todella vaikeaa, ellei lähes mahdotonta, hankkia varsinkin vuosien jälkeen. Näin kävi Space Shuttle ohjelman raketeissa käytetylle viiden prosessorin General Purpose Computer -tietokoneelle. Sitä ei pystytty laitteiston ja ohjelmiston tarkkojen vaatimusten takia päivittämään yli viiteentoista vuoteen. [2]

Edellämainittu osien harvinaisuus tai muuten vaikea saatavuus on osatekijänä myös lento-ohjelmistojen kehitysvaiheiden ongelmissa. Kehityksen aikana on yleistä, että puuttuvat osat tai niiden myöhästynyt toimitus hidastaa testausta tai lopullista toteutusta. Myöhästymiset johtuvat osien vaatimuksista sopimisesta ja myyjän valinnasta, sillä osat tuottaa yleensä ulkopuolinen taho. Osan saapuessa se saattaa olla erilainen kuin odotettiin ja ohjelmisto joudutaan rakentamaan mukailemaan tätä muutosta. Toinen integraatioon liittyvä ongelma on erilaiset testausalustat, jotka vaativat jälleen ohjelmiston integraatiota, mikä ei lisää ohjelmiston luotettavuutta. Testausvaiheessa työryhmän laitteisto saattaa hajota tai niitä ei ole tarvittavaa määrää, jolloin testaus joutuu odottamaan laitteiston vapautumiseen tai uuden hankkimiseen asti. Jos ohjelmisto rakennetaan jonkin sovellusalustan päälle, voivat päivitykset sovellusalustaan muokata käytettyjä osia niin paljon, että ohjelmisto joudutaan integroimaan siihen uudelleen, että se toimisi. [4]

3 Ohjelmistovirheiden mitigointiratkaisut

Monimutkaisten systeemien tutkimuksissa on huomattu, että ohjelmistovirheet voivat olla syynä jopa puolessa tapahtuneista virhetilanteista.[1] [2] [3] [4] Niistä on tullut ajanmittaan suurien systeemien pääasiallinen virheiden lähde. Bugien luokittelua eri tyyppeihin ei tehdä vain teorian takia, vaan sillä on myös käytännön tarkoitus. Erityyppiset bugit ja niiden aiheuttamat häiriöt kehityksen, testauksen ja käytön aikana vaativat erilaisia tapoja niiden käsittelyyn. Myös se, aiheuttaako bugi ohjelman ajon pysähtymisen (failure) vai virhetilan (fault/error), vaikuttaa siihen miten bugi kannattaa käsitellä. [5]

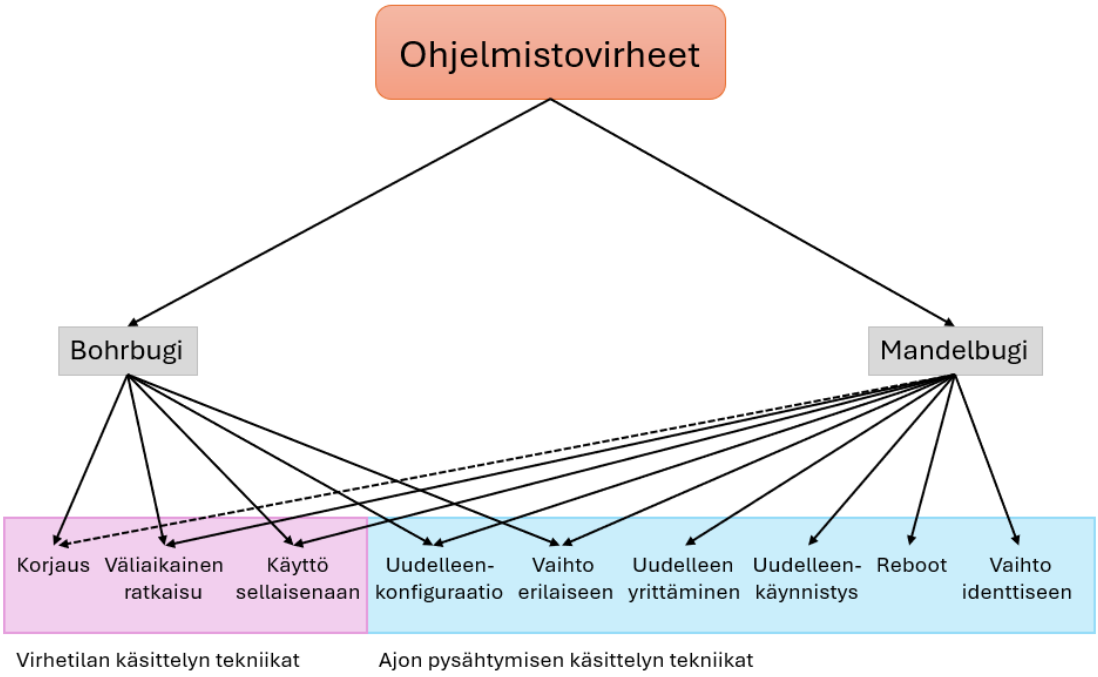
Tarkastellaan ensin miten tutkimuksessa suositellaan ohjelman ajon pysäyttävien bugien käsittelyä. Bohrbugit on yleensä helposti löydettävissä ja korjattavissa testauksen aikana. Usean version ohjelmointi voi estää jäljelle jääneitä bugeja aiheuttamasta vikatiloja, kunhan eri implementoinnit eivät sisällä bugeja jotka aiheutuvat samasta syötteestä. Mandelbugeihin, joita on hankala poistaa testauksenkaan

[1] J. Gray, "Why do computers stop and what can be done about it?", Tandem Computers, Tech. Rep. 85.7, PN87614, 1985.

[2] J. Gray, "A census of Tandem system availability between 1985 and 1990.", *IEEE Trans. Reliability*, vol. 39, no. 4, pp. 409–418, 1990.

[3] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?", *Proc. 4th Conference on USENIX Symposium on Internet Technologies and Systems*, vol. 4, 2003, pp. 1–16.

[4] S. Peret and P. Narasimham, "Causes of failure in web applications," Carnegie Mellon University, Tech. Rep. CMU-PDL-05-109, 2005.



Kuva 3.1: Erilaiset vikatilojen ratkaisumahdollisuudet. Mukailtu lähteestä [5]

aikana, toimii myös usean version ohjelmointi. Kuitenkin tämäntyyppisten bugien ei-deterministisen luonteen vuoksi apua voi olla myös toimenpiteen uudelleen yrittämisestä, identtiseen varaohjelmistoon siirtymisestä ja ohjelman tai tietokoneen uudelleenkäynnistämisestä. AR-bugien tapauksessa ennakoivina toimenpiteinä ovat ohjelmistoa nuorentavat lähestymistavat. Tälläisiä ovat ylläpitotoimenpiteet, jotka siivoavat ohjelman sisäisen tilan ja käynnistää järjestelmän uudelleen sen alkutilaan, jolloin bugin aiheuttamat virhetilat nollaantuvat. [5]

Ohjelman virhetilojen ja ajon pysähtymisen käsittelyyn on tarjolla monia eri vaihtoehtoja (Taulukko 3.1) vikatilasta riippuen (Kuva 3.1). Näistä tunnetuimpia ovat korjaaminen tai paikkaus, väliaikaiset ratkaisut (eng. workaround) tai käyttö sellaisenaan. Koska suurin osa Bohrbugeista käsitellään korjaamalla, voidaan päätellä, että ohjelmistojen ylläpidon kehitys tulee olemaan tärkeä osa lentojen tehtäviä. Myös Mandelbugit, AR-bugit mukaan lukien, käsitellään useimmiten korjaamalla. AR-bugien kohdalla voisi olettaa, että väliaikaiset ratkaisut olisivat kaikkein suosi-

Taulukko 3.1: Vaihtoehdot virheiden käsittelyyn

Tyyppi	**Kuvaus**
Korjaus	Ohjelmistosysteemiin tehdään yksi tai useampi muutos.
Uudelleenkonfigurointi	Systeemin komponentit uudelleenjärjestetään, niitä lisätään tai poistetaan käytettävyyden palauttamiseksi.
Uudelleen yrittäminen	Virhetilan aiheuttanutta komponenttia yritetään käyttää uudelleen ohjelmiston tietystä tilasta eteenpäin.
Uudelleenkäynnistys	Virhetilan aiheuttaneen komponentin uudelleenkäynnistys. Voidaan tehdä ennakoivana toimenpiteenä.
Vaihto identtiseen	Epäonnistunutta laskentaa yritettään uudelleen identtisellä, mutta erillisellä. komponentilla
Vaihto erilaiseen	Epäonnistunutta laskentaa yritettään uudelleen erilaisella komponentilla.
Väliaikainen ratkaisu	Muutetaan muiden komponenttien tapaa vuorovaikuttaa virheen aiheuttaneen komponentin kanssa.
Käyttö sellaisenaan	Jatketaan komponentin käyttöä sellaisenaan virheen syyn ja mahdollisen korjauksen ollessa tuntematon.

tuin tapa käsitellä bugi, kun otetaan huomioon miten vaikeaa tämän tyyppiset bugien alkuperä on löytää. Väliaikaisesta ratkaisusta kuitenkin usein seuraa joidenkin ohjelmien osiin pääsyn menettäminen. Koska ohjelmistojen on usein tarkoitus tehdä hyvin tarkkoja tieteellisiä mittauksia, on tämän tyyppisten virheiden poistaminen korkealla tärkeysjärjestyksessä. Lisäksi aluksien järjestelmät keräävät hyvin tarkat lokit järjestelmän parametreistä, mikä helpottaa bugien alkuperän löytämistä, kun virheeseen johtanut tilanne voidaan toistaa testiympäristössä. Tämä mahdollistaa myöhemmin bugien korjaamisen. [5]

Ohjelmiston ympäristön monimuotoisuus voi lisätä ohjelmiston Mandelbugien sietokykyä. Luonteestaan johtuen ympäristön muutos, esimerkiksi ohjelman uudelleen käynnistäminen, saattaa ratkaista Mandelbugista johtuvan ongelman. Jos Mandelbugi ilmestyy ajanhetkellä t_0 ja sen aiheuttanut ohjelma käynnistetään uudelleen n määrä aikaa myöhemmin ($t_{0+\mathrm{n}}$), saattaa suoritettavana olla esimerkiksi eri ohjelmat kuin ajanhetkellä t_0, jolloin ympäristö on eri, eikä Mandelbugia ilmaannu. Ympäristön monimuotoisuus pyrkii siis luomaan tai pakottamaan uuden, erilaisen

suoritusympäristön ohjelmalle, mutta tarvittaessa se myös siivoaa sisäisen tilan virheitä uudelleenkäynnistyksen, rebootin tai muiden vastaavien toimien avulla. AR-bugien kohdalla näitä toimia voidaan tehdä ennaltaehkäisevästi, mitä kutsutaan ohjelmiston nuorentamiseksi. [6]

Sertifiointi toimivuuden takaamisena

JPL, joka tuottaa suurimman osan NASAn ohjelmistoista rakettien ja luotaimien lisäksi, on ottanut käyttöön erillisen seritifointiprosessin planeettojen välisillä lennoilla käytettäville aluksen hallintaohjelmistoille. Ohjelmistokehittäjien on seurattava koodille laadittua standardia tarkasti, eikä harvoja poikkeuksia lukuun ottamatta ohjeistuksesta poikkeamista hyväksytä. Tämä johtuu siitä, että säännöt perustuvat aiemmilla lennoilla tapahtuneisiin vaaratilanteisiin. Jokainen ohjelman koonitversio tarkastetaan useammalla staattisen analyysin suorittavalla ohjelmalla ja esille nousseet eriävyydet käydään läpi. [7][8]

JPL on luonut myös ohjelmistokehittäjien sertifiointikurssin, jonka läpäisyä vaaditaan, ennen kuin ohjelmistokehittäjä voi työskennellä avaruuslennoilla käytettävän koodin parissa. Kurssi koostuu kolmesta moduulista: tietotekniikan perusteet, JPL:n ohjelmistokehityksen standardien menetelmät ja ohjelmistojen riskit ja heikkoudet. Jokainen moduuli kestää kaksi päivää ja päättyy kokeeseen, joka on läpäistävä. Kurssin tarkoituksena on varmistaa, että tärkeän koodin parissa työskentelevät ovat varmasti tarpeeksi tietoisia tietotekniikan perusteorioista, tavallisimmista algoritmeista, tuntevat hyvin käyttämiensä koodauskielien riskit ja ovat ulkoaopettelun lisäksi myös sisäistäneet JPLn koodausstandarit. [7]

4 Mitigointiratkaisujen implementointi menneisiin lentoihin

Historia on nähnyt monta ohjelmistovirheestä johtuvaa aluksen menetystä. Lähes jokaiselta vuosikymmeneltä aina 60-luvulta lähtien tunnetaan yksi tai useampi tapaus, jossa miljoonien dollarien laitteisto on menetetty yksinkertaisen ohjelmointivirheen vuoksi, mikä puhuu aiemmin todettujen Bohrbugien katastrofaalisten seurausten puolesta. Tässä kappaleessa käymme läpi pari tällaista tapahtumaa ja miten ne oltaisiin voitu välttää erilaisten mitigointitekniikoiden avulla.

4.1 Mars Climate Orbiter ja Mars Polar Lander

Vuonna 1993 NASA aloitti Mars Surveyor -tutkimusohjelman, jonka tarkoituksena oli olla erilaisten Marsia tutkivien tehtävien sarja. Tutkimusohjelmaan kuuluivat Mars Climate Orbiter (MCO) ja Mars Polar Lander (MPL) -avaruusluotaimet, joiden tehtävä oli tutkia planeetan kaasukehää, ilmastoa ja mahdollisuuksia siihen, oliko Marsilla ollut joskus elämälle sopivia olosuhteita. MPLn oli tarkoitus myös etsiä planeetan pinnalta ja maan alta jääesiintymiä. [9] [10]

MCO laukaistiin joulukuussa 1998 ja MPL tammikuussa 1999. Tarkoitus oli, että MCO saapuisi Marsiin yhdeksän kuukautta myöhemmin, käynnistäisi päämoottorinsa elliptiselle kiertoradalle pääsemiseen. Sen jälkeen se hidastaisi vauhtiaan käyttäen aerobraking -tekniikkaa niin, että kiertoradasta tulisi pyöreä. MPL oli tarkoitus las-

keutua Marsin pinnalle kahdesta kolmeen viikkoa sen jälkeen kun MCO oli päässyt halutulle kiertoradalleen. MPL ja kaksi Deep Space 2 anturia oli kiinnitetty yhteiselle alustalle ja anturit oli määrä ampua Marsin pinnan alle noin 60 kilometrin päähän MPLn laskeutumispaikasta. MPL käyttäisi laskuvarjoa hidastamaan vauhtiaan, ennen kuin lähempänä maanpintaa varjo irrotetaan ja laskeutumista hidastavat enää sen pohjaan asennetut raketit. [9]

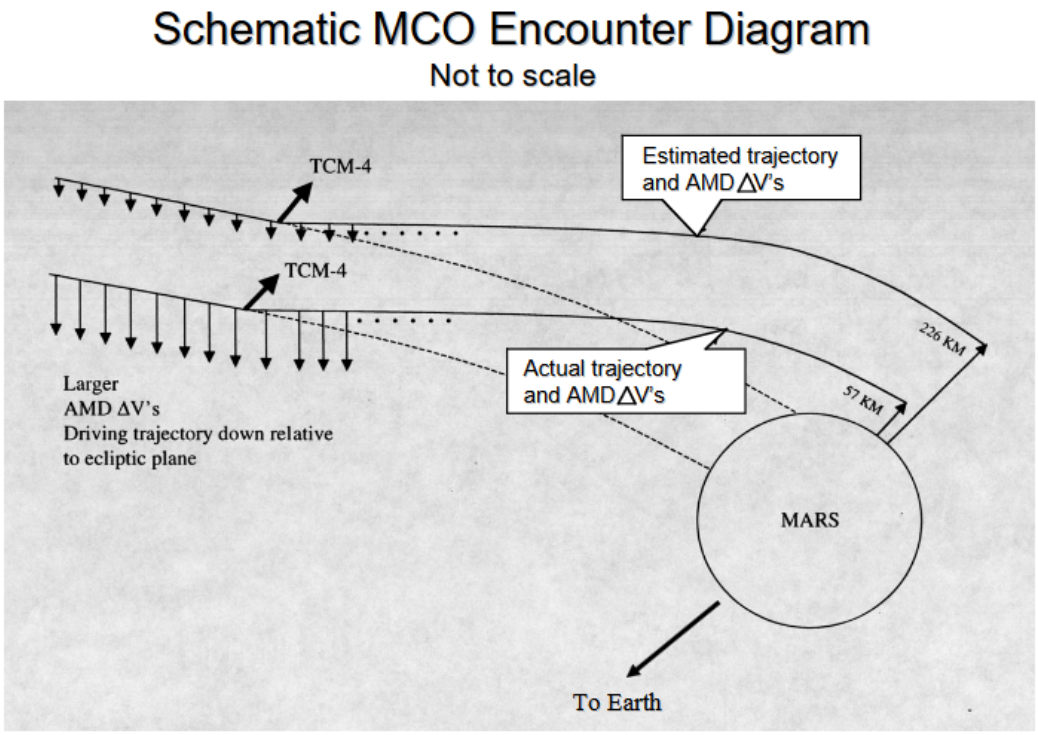
Mars Climate Orbiter

Kaikki oli sujunut Mars Climate Orbiterin kanssa suunnitelmien mukaan siihen asti, kunnes päämoottori käynnistettiin ja MCOn oli tarkoitus alkaa jarruttaa päästäkseen elliptiselle kiertoradalle, jonka periapsis, eli kiertoradan matalin korkeus, olisi 226 km. Pian moottorin käynnistyksen jälkeen luotain menetettiin. [9]

Kevään ja kesän 1999 aikana oli huomattu, että luotaimen radan korjausliikkeistä saatu data erosi odotetusta. Ne oli kuitenkin vain epävirallisesti raportoitu eikä eroavuuksia selvitetty. Kun viimeinen luotaimen epäsymmetrisistä aurinkopaneeleista johtuva radan korjausliike oli suoritettu viikkoa ennen päämoottorin käynnistystä, operaation navigointitiimi laski, että ensimmäisen periapsin etäisyys oli laskenut 150-170 kilometriin. 24 tuntia ennen päämoottorin käynnistystä MCOhon alkoi vaikuttaa Marsin gravitaatio ja dataa sen vaikutuksesta kerättiin tarkempien kiertoratalaskelmien tekemiseksi. Tuntia ennen moottorin käynnistystä laskelmat valmistuivat ja niiden mukaan ensimmäisen periapsin korkeus olisi enää vain 110 km. Minimikorkeus, josta luotain selviäisi oli laskettu olevan 80 km. Luotain kaarsi Marsin taakse 49 sekuntia odotettua aiemmin ja kun sen olisi 21 minuutin jälkeen pitäny ilmestyä jälleen Marsin takaa, luotaimeen ei enää saatu yhteyttä. [9]

Kun luotaimen menetystä alettiin tutkia, tarkasteltiin aiemmin havaittuja korjausliikkeen jälkeisiä nopeuden muutoksen datan eroavuuksia. Niistä ilmeni, että Angular Momentum Desaturation (AMD) -tiedoston sisältämän datan arvot olivat

4,45 kertaisesti liian pieniä. AMDhen oli tallennettu SM_FORCES (small forces, SMF) -ohjelman tulosteita. Ohjelman sisällä käytettiin englantilaisia yksiköitä SI-yksiköiden sijasta, vaikka ohjelman käyttäjäliittymän dokumentoinnissa ja lentoradan laskennassa vaadittiin ohjelman tulostavan SI-yksiköissä. SMF oli aiemmasta Mars Global Surveyor -avaruusluotaimesta löytyneen ohjelmiston muunnelma. Vaikka sen koodissa yksikkömuutos oli tehty, se jäi huomaamatta, sillä se oli piilossa yhtälön sisällä eikä sitä oltu kommentoitu koodiin näkyvästi. Tämä johti suoraan 4,45 kertaisesti vääriin mittaustuloksiin, sillä halutun datan olisi pitänyt olla Newton sekunneissa eikä paunavoimana. Yksi paunavoima on 4,45-kertainen newtoniin verrattuna. Kun periapsis laskettiin uusilla korjatuilla arvoilla ja sillä datalla mitä oli saatu yhteyden menettämiseen asti, oli luotaimen ensimmäisen periapsiksen korkeus vain 57 km (kuva 4.1), mikä oli liian alhainen pitämään luotaimen kiertoradalla. [9]



Kuva 4.1: Mars Climate Orbiterin suunniteltu (ylempi) ja toteutunut lentorata [9]

Koska kyseessä oli Bohrbugi, olisi sen suora korjaaminen suositeltava strategia kun eroavaisuudet huomattiin. Jos mahdollista vian sisältämää koodia oli kuitenkin paljon, saattoi aikaraja tai muut resurssit tulla vastaan. Mikäli kuitenkin ehdittiin huomata että data erosi 4,45-kertaisesti halutusta, oltaisiin koodiin voitu tehdä lisäys joka kertoo datan 4,45 :llä ennen sen tallennusta tai käyttöä. Mikäli ongelma huomattaisiin vasta sellaisessa vaiheessa, ettei koodia pystytä enää päivittämään, olisi vaihtoehtona käyttää ohjelmaa sellaisenaan, mutta huomioida virhe radan korjausliikkeitä suunniteltaessa.

Mars Polar Lander

Kuten MCOkin, MPL oli selvinnyt Maan ja Marsin välisestä matkasta ilman isompia ongelmia. 6,5 tuntia ennen Marsin kaasukehään saapumista oli suoritettu viimeinen lentoradan korjaus. Luotain käännettiin laskeutumisasentoonsa, mikä tarkoitti odotettua yhteyden menetystä, sillä MPLn antenni osoitti tällöin poispäin Maasta. Tarkoitus oli, että MPL aloittaisi 45-minuuttisen datan lähetyksen 24 minuuttia laskeutumisensa jälkeen ja anturieista saapuisi dataa noin seitsemän tunnin kuluttua. Yhteyttä kumpaankaan ei kuitenkaan saatu. [10]

MPL oli kehitysvaiheessaan saanut huomattavasti pienemmän budjetin verrattuna aiempiin Mars-luotaimiin ja tästä syystä kehityksessä jouduttiin turvautumaan toimiin, jotka altistivat projektin monille ongelmille ja virheille. Koska NASAn Jet Propulsion Laboratory (JPL) pystyi kiinnittämään projektiin vain kymmenhenkisen työryhmän, turvautuivat he vahvasti luotainten pääurakoitsijan Lockheed Martin Astronauticsin (LMA) työnjohto- ja insinöörirakenteeseen. JPLn työntekijöitä toimi vain ylemmän tason valvontatehtävissä ja JPLn teknisten eksperttien osallisuus oli hyvin pientä. LMAn työntekijät tekivät usein 60-, jopa 80-tuntisia työviikkoja pitkien ajanjaksojen ajan projektin parissa. Pieni budjetti johti myös siihen, että tärkeistä teknisistä asioista vastasi vain yksi työntekijä ja puutteellinen vuorovaiku-

tus muiden työntekijöiden kanssa oli suuri ongelma. Tämä puolestaan johti siihen, ettei projektin yhteydessä ehditty pohtia tehtyjen päätösten seurauksia eikä työntekijöillä ollut aikaa tai tarpeeksi henkilöstöä tekemään saman verran testauksia kuin JPLn projekteissa yleensä. [10]

Varmaa syytä MPLn menetykselle ei koskaan löydetty. Tapahtumaa tutkimaan valittu komitea kuitenkin arvioi todennäköisimmäksi syyksi ohjelmistovirheen. Jokaisessa MPLn jalassa oli magneettinen sensori, jonka oli tarkoitus aistia luotaimen laskeutuminen, jonka jälkeen laskeutumismoottorit sammutettaisiin. Useissa testeissä huomattiin, että laskeutumisjalkojen aktivoituessa magneettiset sensorit huomasivat laskeutumissignaalin tyyppistä aktiivisuutta. Nämä laskettiin oikeiksi lukemiksi ohjelman puolesta, jos signaalit jatkuivat tarpeeksi pitkään, mitä useissa testeissä tapahtui. Ohjelman oli tarkoitus jättää huomiotta tämän tyyppiset signaalit ennen tiettyä laskeutumislogiikan käynnistystä, mutta tätä ohjelman osaa ei oltu kunnolla implementoitu. Tästä johtuen ohjelma luultavasti sammutti laskeutumismoottorit 40 metrin korkeudessa Marsin pinnasta, kun laskeutumislogiikka käynnistettiin. MPLn oli tarkoitus osua Marsin pinnalle 2,4 m/s vauhdilla, mutta ilman moottoreiden hidastusta vauhti on ollut suurin piirtein kymmenkertainen, mikä olisi hajottanut luotaimen. [10]

Kyseessä on jälleen suoraviivaisesti ohjelman koodauksesta johtuva virhe. Ottaen huomioon luotaimen suunnittelun ongelmat, yhtä suoraa ongelman ratkaisevaa tapaa on vaikea keksiä. Paras tapa korjata ongelma olisi luultavasti ollut estää moottoreiden sammuttavan koodin ajo ennen tiettyä pistettä, mitä oltiin epäonnistuneesti yritetty. Olettaen että sitä ajanjaksoa, jona jaloista tuleva laskeutumissignaali oli aktiivisena, ei voitu pidentää ilman, että raketti vaurioutuisi, olisi muut vaihtoehdot muuttaa ohjelmiston toimintaa tai luotaimen varustelua. Koska laskeutumisjaloista saatava signaali ei ollut luotettava, olisi moottorien sammutus pitänyt liittää johonkin muuhun attribuuttiin, kuten aikaan tai korkeuteen. Jos luotaimessa ei ollut

korkeusantureita, oltaisiin voitu tehdä laskelmat siitä, miten korkealta luotain voisi pudota turvallisesti ilman raketteja ja milloin tämä korkeus alitettaisiin lentorata huomioon ottaen.

4.2 Ariane 5 lento 501

Ariane 5 oli Euroopan avaruusjärjestön kehittämä kertakäyttöinen kantoraketti, jonka eri versiot olivat käytössä vuodesta 1996 aina kesään 2023 asti. Se pystyi kuljettamaan huomattavasti enemmän lastia kuin edeltäjänsä Ariane 4 ja olikin Euroopan avaruusjärjestön pääasiallinen tapa viedä luotaimia avaruuteen. [11]

Rakettia käytettiin ensimmäistä kertaa kesäkuussa 1996 ja se päättyi raketin tuhoutumiseen pian laukaisun jälkeen, kun raketti suistui suunnitellulta lentoradaltaan ja räjähti. Kaikki oli mennyt suunnitelmien mukaan raketin laukaisuun asti, poislukien tunnin viivästystä johtuen sääoloista. 37 sekuntia laukaisun jälkeen varainertiasuunnistusjärjestelmä petti, mitä seurasi saman tien aktiivisena olleen intertiasuunnistusjärjestelmän (SRI, eng. Inertial Reference System) pettäminen. Tämä aiheutti raketin kurssin yhtäkkisen muutoksen. Suuri nopeus ja raketin kääntymisestä johtuva ilmanvastus sen sivulle aiheutti booster-rakettien kiinnityksen repeämisen. Raketin hallinnan menetys aktivoi aluksen itsetuhomekanismin ja raketti räjähti ilmakehässä. [12]

Ongelman aiheuttanutta inertiasuunnistusjärjestelmää, ja varsinkin sen ohjelmistoa, oli aiemmin käytetty Ariane 4 -raketeissa ilman ongelmia. Sitä käytettiin raketin laukaisualustalla tapahtuvassa raketin kohdistusprosessissa, eikä siis ollut tarpeellinen lentoon nousun jälkeen. Ohjelman annettiin kuitenkin olla päällä hetken lentoon nousun jälkeenkin, sillä joskus Ariane 4 :n lento saattoi viivästyä joitain sekunteja ja tällä vältettiin tarve ohjelmiston uudelleenajolle. Tarvetta tälle ei ollut Ariane 5 :n kanssa. SRI lähettää dataa lentotietokoneelle, joka ajaa lento-ohjelmistoa

ja ohjaa rakettien suuttimia. Jos aktiivinen SRI lakkaa toimimasta, lentotietokone vaihtaa vara-SRI :hin, mikäli vara-SRI toimii oikein. [12]

Ongelmat aloitti Ariane 5 :n lentoradan erilainen alku. Alussa sen lentorata on enemmän kallellaan kuin Ariane 4 :n, mikä yhdistettynä Ariane 5 :n suurempaan nopeuteen aiheutti suuremman horisontaalisen nopeuden. Tätä laski SRIn ohjelmassa sisäinen kohdistusfunktio, jonka tulos tallennettiin Horizontal Bias -muuttujaan. Lentotietokone vaati luvun etumerkillisenä 16-bittisenä integer -lukuna. Horizontal Bias -muuttujaan tallennettu 64-bittinen liukuluku oli kuitenkin isompi kuin mitä 16-bittisellä etumerkillisellä integer -luvulla pystyi esittämään ja tämä aiheutti Operand Error -virheen SRIssä. Tällaiseen ylivuotovirheeseen ei ollut koodissa valmista käsittelyä ja se aiheutti SRIn toiminnan keskeytymisen. [12]

Lentotietokone ei myöskään pystynyt vaihtamaan vara-SRIhin, sillä se oli lakannut toimimasta edellisen datasyklin aikana juuri samasta syystä. Aktiivisen SRIn lentotietokoneelle lähettämä data sisälsi virheen ilmaannutua muutakin kuin lentodataa, mutta lentotietokone tulkitsi sen lentodataksi. Tästä johtuen se käänsi raketin pois lentoradalta. [12]

Jälleen virheen aiheutti koodista löytyvä Bohrbugi. Käytössä oli mitigointistrategia, missä ajon pysäyttänyt komponentti vaihdettiin identtiseen komponenttiin, mikäli SRI kohtaisi toiminnan keskeyttävän virheen. Koska virhe kuitenkin löytyi molemmista laitteistoista, olisi ollut parempi, jos vara-SRI olisi ollut erilainen aktiiviseen verrattuna. Myös systeemin uudelleenkonfigurointi voisi auttaa.

5 Yhteenveto

Tutkielmassa pyrittiin selvittämään mikä hankaloittaa avaruudessa toimiviin laitteistoihin tulevien ohjelmistojen suunnittelua, millaisia virheitä niistä löytyy ja miten niitä tulisi käsitellä. Tutkimuksista nähtiin, että ohjelmistojen suunnittelua vaikeutti suuressa osin avaruuden tuottamat ongelmat, kuten säteily, tila- ja painorajoitukset sekä suuret välimatkat. Tämän vuoksi tarvitaan spesifisti suunniteltuja laitteistoja ja monitieteellisiä suunnitteluryhmiä, joiden kommunikaatiovirheet voivat aiheuttaa suuriakin ongelmia aluksen valmistumisen jälkeen.

Eniten ohjelmistoissa oli katastrofaalisia seurauksia aiheuttavia Bohrbugeja ja toiseksi eniten Mandelbugeja, joiden seuraukset olivat mitättömiä. Virhetyypin ja sen aiheuttaman riskin kriittisyydellä[1] ei ole yhteyttä. Bugin aiheuttavan koodin korjaamisen lisäksi on olemassa useita eri virheiden käsittelytapoja, joita voidaan käyttää mikäli ohjelmakoodin muuttaminen ei ole mielekästä tai bugin aiheuttavaa koodinpätkää ei tiedetä. Näitä ovat muun muassa väliaikaisen ratkaisun käyttö, vaihto identtiseen tai erilaiseen ohjelmistoon sekä uudelleenkäynnistys.

Jatkotutkimuksena olisi hyödyllistä selvittää, miten avaruuslaitteistoiden suunnittelua tulisi muuttaa bugimäärien vähentämiseksi. Hyviä tutkimuskohteita voisi olla erilaiset testausstrategiat, niiden optimointi ja miten mahdollisimman realistinen testausympäristö voitaisiin toteuttaa.

[1]Kriittisyydellä tarkoitetaan tässä sitä, kuinka vakavan riskin bugi aiheuttaa. Tutkimuksessa virheiden riskit luokiteltiin neljään eri vakavuusluokkaan: ei hyväksyttävä riski, hyväksyttävä riski, ei huomattavaa riskiä tai ei riskiä ollenkaan.

Lähdeluettelo

[1] M. Grottke, A. P. Nikora ja K. S. Trivedi, "An empirical investigation of fault types in space mission system software", teoksessa *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 2010, s. 447–456. DOI: 10.1109/DSN.2010.5544284.

[2] C. W. Johnson, "The natural history of bugs: Using formal methods to analyse software related failures in space missions", teoksessa *International Symposium on Formal Methods*, Springer, 2005, s. 9–25.

[3] J. E. Tomayko, "Computers in Spaceflight - The NASA Experience", *NASA Contractor Report 182505*, 1988.

[4] H. Sukhwani, J. Alonso, K. S. Trivedi ja I. Mcginnis, "Software Reliability Analysis of NASA Space Flight Software: A Practical Experience", teoksessa *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2016, s. 386–397. DOI: 10.1109/QRS.2016.50.

[5] J. Alonso, M. Grottke, A. P. Nikora ja K. S. Trivedi, "An empirical investigation of fault repairs and mitigations in space mission system software", teoksessa *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, s. 1–8. DOI: 10.1109/DSN.2013.6575355.

[6] K. S. Trivedi, M. Grottke ja J. A. Lopez, "Rethinking Software Fault Tolerance", *IEEE Transactions on Reliability*, vol. 73, nro 1, s. 67–72, 2024. DOI: 10.1109/TR.2023.3330787.

[7] K. Havelund ja G. J. Holzmann, ”Software certification - coding, code, and coders”, teoksessa *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, 2011, s. 205–210.

[8] G. J. Holzmann, ”Landing a Spacecraft on Mars”, *IEEE Software*, vol. 30, nro 2, s. 83–86, 2013. DOI: 10.1109/MS.2013.32.

[9] A. G. Stephenson, D. R. Mulville, F. H. Bauer et al., *Mars Climate Orbiter Mishap Investigation Board Phase I Report*. Jet Propulsion Laboratory, 1999.

[10] A. Albee, C. Leising, S. Battel et al., *Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions*. Jet Propulsion Laboratory, 2000. url: https://ntrs.nasa.gov/citations/20000061966.

[11] *Ariane 5*, 2024. url: https://www.esa.int/Enabling_Support/Space_Transportation/Launch_vehicles/Ariane_5, (Viitattu 11.3.2024).

[12] J.-L. Lions, L. Lübeck, J.-L. Fauquembergue et al., *ARIANE 5 Flight 501 Failure Report by the Inquiry Board*, 1996.