SeAMK

SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Tuomas Männistö

Ohjausjärjestelmien testaus Epecillä

Nykytila ja kehityskohteet

Opinnäytetyö
Syksy 2020
SeAMK Tekniikka
Automaatiotekniikka

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: SeAMK Tekniikka

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Tuomas Männistö

Työn nimi: Ohjausjärjestelmien testaus Epecillä. Nykytila ja kehityskohteet

Ohjaaja: Marko Hietamäki

Vuosi: 2020 Sivumäärä: 33

Tämän opinnäytetyön tavoitteena oli selvittää Epec Oy:lle ohjausjärjestelmien testauksen nykytilaa ja mahdollisia kehityskohteita yrityksen Engineering Services -osaston osalta.

Opinnäytetyössä avataan käytössä olevia menetelmiä, ohjelmistoja sekä tapoja ja tarkastellaan yhden esimerkkinä toimivan asiakasprojektin kautta minkälaisiin tuloksiin nykymenetelmillä voidaan päästä. Esimerkkiprojektista saatujen tulosten perusteella työssä analysoidaan, mihin tulevaisuudessa tulee kiinnittää huomiota ja esitetään mahdollisia kehityskohteita.

[1] Asiasanat: ohjausjärjestelmä, testaus, testausmenetelmät, laadunvarmistus

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: SeAMK Technology

Degree programme: Automation Engineering

Specialisation: Electrical Automation

Author: Tuomas Männistö

Title of thesis: Testing of Control Systems at Epec. Current Status and Development Targets

Supervisor: Marko Hietamäki

Year: 2020 Number of pages: 33

The aim of the thesis was to study the current state of control systems' testing and to find potential development targets for the Engineering Services department of Epec Oy.

The thesis studied the used methods, software and the process of testing and examined what kind of results could be achieved through one client project that served as an example. Based on the results of the example project, the work analyzed the potential development targets to which attention should be paid in the future.

[1] Keywords: control system, testing, testing methods, quality assurance

SISÄLTÖ

Kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

ALM	Sovelluksen elinkaaren hallinta
Analyysi	Tutkittavana olevan asian määrittäminen.
Black-box-testaus	Testausmenetelmä, jossa testaus suoritetaan ilman tietoa järjestelmän sisäisestä toiminnasta.
Codesys	Epecillä käytössä oleva ohjelmointityökalu, jonka on kehittänyt Saksalainen CODESYS GmbH.
HIL	Hardware In the Loop, ulkoinen ohjainyksikkö liitettynä simulaattoriin, joka simuloi laitteistoin toimintaa.
Integraatiotestaus	Ohjelmiston komponenttien yhteentoimivuuden testaamismenetelmä.
IoT	IoT eli Internet of Things, esineiden internet, tarkoittaa laitteiden liittämistä internetverkkoon.
I/O	Input / Output eli tulo ja lähtö, ovat järjestelmän ulkoisista liitännöistä yleisesti käytetyt termit.
Ohjausyksikkö	Laite, jolla ohjataan koneen sähköjärjestelmää tai osajärjestelmää.
SIL	Software In The Loop, ohjausyksiköt on simuloitu ja liitetty simulaattoriin joka simuloi laitteiston toimintaa.
TestInterface	Käyttöliittymä jonka avulla voidaan muuttaa virtuaalisten ohjainyksiköiden I/O:ten tiloja.
Testitapaus	Yksittäinen tilanne testattavassa järjestelmässä.
VCU	Virtuaalinen ohjainyksikkö.
VTE	Virtuaalinen testausympäristö, on Epecillä käytössä oleva SIL.

White-box-testaus	Testaustapa, jossa testaus suoritetaan tarkastelemalla järjestelmän sisäistä toimintaa.
Yksikkötestaus	Yksittäisen komponentin tai funktion testausmenetelmä.

1 Johdanto

1.1 Työn tausta

Epec Oy valmistaa ohjausyksikköjä ja niiden ohjelmistoja. Ohjelmistojen laadunvarmistuksessa testaus on tärkeässä osassa. Testauksen kehittämistä varten tarvittiin selvitys testauksen nykytilasta ja mahdollisista parantamiskohteista.

1.2 Työn tavoite

Työn tavoitteena on muodostaa ymmärrys testauksen nykytilasta ja parantamiskohteista Epec Oy:n Engineering Services -osastolla. Testausmenetelmät, -ohjelmistot sekä -laitteet kehittyvät jatkuvasti. Tässä työssä on tarkoitus kartoittaa nykytilannetta ja analysoida sitä sekä selvittää parantamiskohteita. Tavoite on kerätä tietoa testauksesta niin, että sen pohjalta saadaan tehtyä yleistä ohjeistusta projektien testauksen suunnittelun pohjaksi.

1.3 Työn rakenne

Työssä käsitellään Epecillä tällä hetkellä käytössä olevia ohjelmistoja, laitteita, menetelmiä ja vaiheita. Sen jälkeen analysoidaan miten asiakasprojektissa on onnistuttu laadunvarmistuksessa nykyisin menetelmin. Näiden pohjalta pohditaan, miten testausta tulisi kehittää Epecillä ja pohditaan yleistä suuntaa mihin testaus on kehittymässä. Lopuksi analysoidaan työssä ilmenneitä seikkoja ja tehdään yhteenveto.

1.4 Yritysesittely

Epec Oy on vuonna 1978 Seinäjoelle perustettu yritys. Yrityksen perustaja on Veikko Rintamäki. Yrityksen perustamisesta vuoteen 1989 asti Epec käytti nimeä E-P Elektroniikka. Tuolloin yritys valmisti erilaisia elektroniikkatuotteita, kuten suuria ulkoilmanäyttöjä. Lisäksi yritys teki elektroniikkasuunnittelua ja myi tietokoneita. Nykyisin Epec on elektroniikka-alan toimija, joka on erikoistunut liikkuvien työkoneiden koneenohjausjärjestelmiin ja sen komponentteihin. Vuonna 2004 Epecistä tuli osa Ponsse Oyj:n konsernia, kun metsäkoneyhtiö Ponsse osti osana kasvustrategiaansa Epecin. (Epec, [viitattu 15.2.2021].)

Epec valmistaa ja suunnittelee sulautettuja näyttöjä ja ohjausyksiköitä sekä IoT-laitteita. Niiden lisäksi Epec Oy tarjoaa palveluita asiakasprojektien suunnitteluun ja toteutukseen, asiakastukeen sekä ohjelmistokoulutukseen. (Epec, [viitattu 15.2.2021].)

Epecin asiakaskunnassa on muun muassa maatalouskoneiden, metsäkoneiden, työstökoneiden ja kaatopaikkakoneiden valmistajia. Epecin asiakkaita ovat muuan muassa Ponsse, Sandvik Mining and Construction ja Metso Minerals. Yrityksen pääkonttori sijaitsee Seinäjoella, jossa on myös yrityksen tehdas. Tehtaassa valmistetaan kaikki Epecin tuotteet. Samoissa tiloissa on myös insinööripalvelut, huolto, tuotekehitys ja asiakastuki. Epecillä on myös toimipisteet Tampereella sekä Kiinassa Shanghaissa. Shanghain toimipisteellä sijaitsee tekninen tuki, joka on pääasiassa Aasian markkinoille. Myös Tampereen toimipisteellä on yrityksen tuotekehitysyksikkö. Epecillä työskentelee tällä hetkellä yli 130 ammattilaista. (Epec, [viitattu 15.2.2021].)

2 Ohjausjärjestelmät ja testaus yleisesti

Testaus on tärkeä osa laadunvarmistusta kaikessa teollisessa tuotannossa. Sitä käytetään tuotteiden ja palveluiden kehitysprosessien jokaisessa vaiheessa määrittelystä lopputuotantoon. Erilaisille valmiille sekä kehityksen eri vaiheissa oleville tuotteille ja palveluille tarvitaan erilaisia testaustekniikoita ja menetelmiä. Ohjausjärjestelmiä on nykyään lähes jokaisessa koneessa. Autot, metsäkoneet, työkoneet yms. toimivat pitkälti ohjausjärjestelmien varassa ja monet näiden koneiden uusista ominaisuuksista on kehitetty ohjausjärjestelmiä kehittämällä. Tämä on johtanut ohjausjärjestelmien laajentumiseen ja monimutkaistumiseen. Samalla myös ohjausjärjestelmien laadunvarmistuksen ja testauksen tarve on kasvanut.

Ohjausjärjestelmät ovat automaatiojärjestelmiä, jotka ohjaavat muita järjestelmiä ja laitteita. Ne ottavat sisääntuloina mittaus- ja ohjausdataa, joiden perusteella lasketaan ulostulot. Ulostulot toimivat muiden järjestelmien ohjauksina. Ohjausjärjestelmät koostuvat laitteistosta ja ohjelmistosta. Liikkuvissa työkoneissa laitteistona toimivat usein ohjausyksiköt ja CAN-väylät, joita pitkin kulkevat mittaus- ja ohjausdata ohjausyksikön ja muun järjestelmän välillä. Ohjelmisto määrää miten ohjausjärjestelmä toimii. Nousevan trendinä on myös koneiden lisääntyvät IoT-ominaisuudet. (Gartner 29.8.2019.) Epec IoT ja connectivity -tuotteet tarjoaa muun muassa ratkaisuja etäkunnossapitoon, diagnostiikkaan, kaluston hallintaan, keskeisten suorituskyky indikaattorien seurantaan tai räätälöityihin asiakastarpeisiin.

3 Testauksen nykytilanne

3.1 Testausympäristöt

Tässä luvussa esitetään kolme erilaista testausympäristöä. Testausympäristöjä on myös muunlaisia, mutta tässä luvussa olevat ympäristöt ovat Engineering Services -osastolla käytössä. Ensimmäiseksi esitellään HIL ja SIL -ympäristöt yleisesti ja sen jälkeen VTE, joka on Epecin oma SIL -ympäristö.

3.1.1 HIL

’Hardware-In-the-Loop-testaus’ tai HIL-testaus on yksi dynaamisen testauksen menetelmä. Siinä ohjausjärjestelmä liitetään kiinni simulaattoriin, joka simuloi laitteiston toimintaa. Ohjausjärjestelmä siis luulee olevansa kiinnitettynä oikeaan ohjattavaan laitteistoon, ja näin päästään hyvin lähelle kohdeympäristöä. Simulaatioiden laatu ratkaisee, kuinka hyvin ohjelmisto lopulta toimii kohdelaitteessa. Tämä mahdollistaa ohjausjärjestelmän testaamisen ennen varsinaisen laitteiston kehittämistä. HIL-simulaattoreita ja testauslaitteistoja on käytössä työkoneita valmistavissa yrityksissä ja autoteollisuudessa sekä ohjausjärjestelmiä kehittävillä yrityksillä. Hyvin usein testattava ohjausjärjestelmä vaatii oman spesialisoidun HIL -ympäristön johdotuksineen ja sopivine simulaatioineen, ja on näin hyvin yleinen osa uuden kehitysprojektin alkuvaihetta. HIL-simulaattoreiden huono puoli on, että ne ovat kalliita joten niitä voidaan rakentaa vain rajallinen määrä. (Bringmann & Krämer 2008, 487.)

3.1.2 SIL

’Software-in-the-loop-testaus’ tai SIL-testaus on termi, jota käytetään kun puhutaan testauksesta, missä testausympäristöä ajetaan kokonaan virtualisoidussa ympäristössä. Yleensä se sisältää myös mallinetun ympäristön ja testattavan laitteen. SIL-simulaattoreissa hyvää on, että ne ovat halpoja toisin kuin HIL-simulaattorit. SIL-käyttö on myös usein helppoa, joten niitä voidaan hankkia vaikka kaikille testaajille ja ohjelmistokehittäjille. SIL-simulaattoreiden huono puoli on, että sen ajoitukset eivät vastaa oikeaa laitetta virtualisointi

ei myöskään vastaa täydellisesti oikeita komponentteja sekä suorituskyky on myös rajallista. (Bringmann & Krämer 2008, 487.)

3.1.3 VTE

VTE eli virtuaalinen testausympäristö, on Epecillä käytössä oleva SIL. VTE-ympäristöön on mahdollista liittää oikeita laitteita CAN-rajapintojen kautta ja jopa IO:ta erillisten IO-korttien kautta. Testausympäristö koostuu kahdesta komponentista EpecTestInterfacesta ja virtuaalisista ohjainyksiköstä (VCU). TestInterface on järjestelmän käyttöliittymä, jonka avulla voidaan muuttaa virtuaalisten ohjainyksiköiden I/O:ten tiloja. VTE-järjestelmään voidaan liittää maksimissaan kymmenen virtuaalista CAN-ohjainyksikköä. (Epec, [viitattu 10.11.2020].)

Virtuaalinen ohjainyksikkö on tietokoneella ajettava SoftPLC-sovellus, joka on tehty simuloimaan Epecillä valmistettavia CAN-ohjainyksiköitä. Ohjainyksikköön voidaan ladata CoDeSySsillä ohjelmoitu PLC-sovellus, joka tallentuu virtuaalisen ohjainyksikön muistiin. Sovelluksen lataus virtuaaliseen ohjainyksikköön tapahtuu TCP/IP-tietoliikenneprotokollan kautta. Järjestelmään on mahdollista kytkeä myös todellisia CAN-toimilaitteita. (Epec, [viitattu 10.11.2020].)

VTE-järjestelmään on rakennettu Python-rajapinta. Rajapinta sisältää Pythonissa käytettäviä komentoja, joiden avulla virtuaalisten ohjainyksiköiden I/O-pinnien tiloja voidaan lukea ja asettaa, sekä lähettää ja vastaanottaa CAN-viestejä. Python-rajapinta mahdollistaa myös testiautomaation toteuttamisen. (Epec, [viitattu 10.11.2020].)

3.2 ALM

Tässä luvussa esitellään kolme erilaista ALM -työkalua. ALM lyhenne tulee sanoista Application Lifecycle Management ja tarkoittaa suomeksi sovelluksen elinkaaren hallintaa. ALM -työkalut sisältävät tyypillisesti ainakin vaatimusten hallinnan, ohjelmistoarkkitehtuurin, kehittämisen, testaamisen, ylläpidon, muutoksen hallinnan, projektin hallinnan ja päivitysten hallinnan.

3.2.1 Test manager (nykyisin Azure test plans)

Azure Test Plans tai Test hub Azure DevOps Server on Azure DevOpsin lisäosa, joka tarjoaa testauksen hallinnan kolme pääelementtiä eli testisuunnitelmat, testi-suitet ja testitapaukset. Niitä voidaan hallita projektien sisällä ja jakaa tiimin kesken. Samalla hyödytään integraatiosta Azure DevOps -taskin kanssa**,** tällä on iso rooli testien ja tulosten jäljitettävyyden kannalta. Integraation ansiosta tulokset voidaan linkittää vaatimuksiin, ja näin pystytään seuraamaan tuotteen valmiusastetta ja laatua. (Azure Test Plans Microsoft, [viitattu 12.11.2020].)

3.2.2 Polarion

Polarion Requirements Management ja Polarion ALM ovat Polarion Softwaren kehittämiä web-pohjaisia ratkaisuja. Polarion tarjoaa yhtenäisen ja kustannustehokkaan web-pohjaisen ratkaisun ohjelmiston elinkaarenhallintaan. Polarion tukee ominaisuuksillaan ohjelmistokehityksen jäljitettävyyttä sekä läpinäkyvyyttä ja tukee kaikkia ohjelmistokehityksen tärkeimpiä vaiheita. (About Polarion Software n.d, [viitattu 12.11.2020].)

3.2.3 Jira tyyppiset ohjelmistot

Jira on Atlassianin kehittämä tehtävienhallintaohjelmisto. Jiran tehtävänhallintatyökalua käytetään projektien hallintaan sekä työmääräysten, virheiden ja tukipyyntöjen raportointiin. Jira tukee myös testitapausten määrittelyä ja linkityksiä vaatimuksiin. (Jira, [viitattu 12.11.2020].)

3.3 Testausmenetelmät

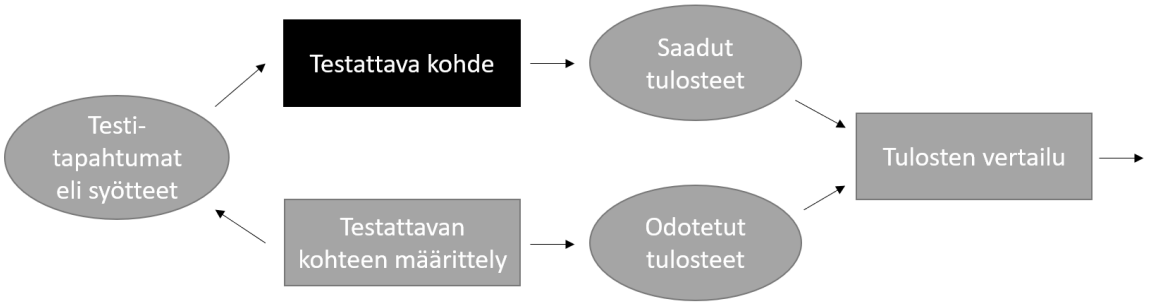
Tässä luvussa kerrotaan erilaisista testausmenetelmistä. Testausmenetelmiä on useita erilaisia ja oikea menetelmä tulee valita projektin ja testattavan kohteen mukaan. Myös testaukseen käytettävät resurssit voivat vaikuttaa testausmenetelmän valintaan.

3.3.1 Ohjelmiston määrittelyä tai vaatimuksia vasten testaaminen

Testaus ohjelmiston tai järjestelmän määrittelyä ja vaatimuksia vasten edellyttää, että saatavilla on laadukkaat ja yksityiskohtaiset spesifikaatiot. Kun vaatimuksia vasten testaan, ei välttämättä määritellä ajettavia testitapauksia, tällöin testaajalla on suuri vastuu ja testaajan taidot vaikuttavat paljon onnistumiseen. (Myers, Sandler & Badgett 2012, 119.)

3.3.2 Black-box

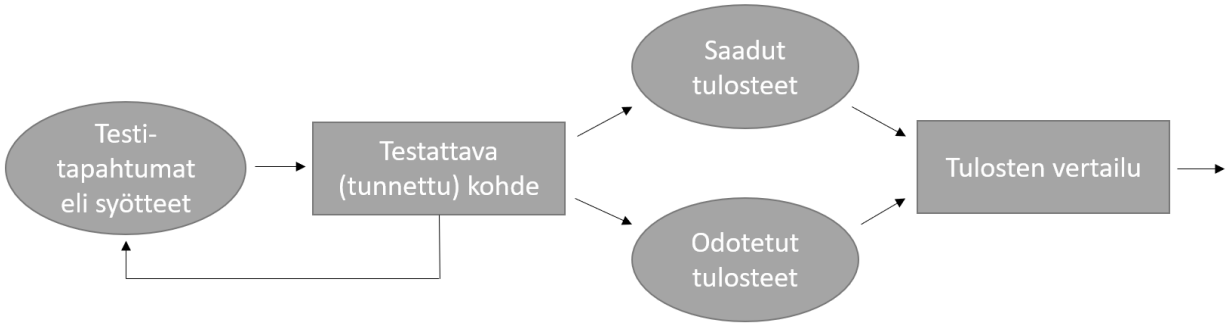
Black-box-testaus perustuu testattavan järjestelmän tai komponentin input-output-käyttäytymiseen. Black-box-testauksessa ei välitetä tai ei ole saatavilla tietoa testattavan kohteen sisäisestä toiminnallisuudesta, vaan tutkittavana ovat kohteen tulosteet (output) erilaisilla syötearvoilla (input). Testaajalle kohde on kuin "musta laatikko" eli black-box. Testattavan kohteen oikeellisuus todetaan vertaamalla saatuja tuloksia haluttuihin jai odotettuihin tuloksiin. Testitapaukset luodaan aina kohteen määrittelyn perusteella. Kuviossa 1 on esitetty black-box-testauksen prosessi. (Myers, Sandler & Badgett 2012, 22.)



Kuvio 1. Black-box-testaus

3.3.3 White-box

Toisin kuin black-box-testauksessa, jossa testitapaukset johdetaan kohteen määrittelystä, white-box-testauksessa testitapaukset johdetaan kohteen, esimerkiksi koko ohjelman sisäisestä rakenteesta ja logiikasta. Testitapaukset luodaan siten, että kohteen kaikki haarat ja ohjelmapolut tulee käytyä läpi. Kuviossa 2 on esitetty white-box-testauksen prosessi. (Myers, Sandler & Badgett 2012, 24.)



Kuvio 2. White-box-testaus

3.3.4 Tutkiva testaaminen

Tutkivassa testaamisessa testattavaa kohdetta käytetään ja tehdään siitä havaintoja sekä yritetään oppia ja ymmärtää sitä. Tutkivassa testauksessa on tärkeää tunnistaa riskialttiita ja testaamista edellyttäviä alueita. Testit tehdään ilman formaaleja testitapauksia tai luodaan niille testauksen yhteydessä testitapaukset ja suoritetaan ne. (Guru99, [viitattu 20.11.2020])

3.4 Testausvaiheet

Tässä luvussa kerrotaan testausvaiheista. Testausvaiheet ovat yleensä liitoksissa vastaavaan suunnittelutasoon. Testausvaiheiden nimeäminen on myös hieman vaihtelevaa eri yritysten välillä.

3.4.1 Yksikkötestaus

Yksikkötestauksessa varmistetaan testattavien ohjelmistoelementtien toiminnan erillään muusta ohjelmasta. Kontekstista riippuen ne voivat olla yksittäisiä aliohjelmia tai suurempia komponentteja, jotka on tehty useasta yhtenäisestä yksiköstä. Tyypillisesti yksikkötestaus tapahtuu niin, että koodiin päästään käsiksi ja tukena käytetään virheenkorjaustyökaluja. Yleensä ohjelmoija joka koodin kirjoittaa, suorittaa sille myös yksikkötestauksen. (Bourque, Fairley 2014, 77). Tyypillisesti yksikkötestauksessa käytetään white-box-tekniikkaa.

3.4.2 Integraatiotestaus

Integraatiotestaus on kaiken tyyppistä testaamista, jossa pyritään todentamaan rajapintoja ohjelmistoelementtien välillä. Ohjelmistoelementit voidaan integroida iteratiivisesti tai kaikki yhdellä kertaa. Integraatiotestauksessa käytetään pääsääntöisesti black-box tekniikkaa, mutta voidaan käyttää myös muitakin tekniikoita. (Guru99, [viitattu 15.2.2021].)

3.4.3 Systeemi integraatio testaus

Systeemi integraatiotestaus on integroidussa laitteisto- ja ohjelmistoympäristössä suoritettavan ohjelmistotestauksen tapa, jolla varmistetaan koko järjestelmän käyttäytyminen. Testaus suoritetaan täydellisellä, integroidulla järjestelmällä, jotta voidaan arvioida onko järjestelmä vaatimuksien mukainen. (Guru99, [viitattu 15.2.2021].)

3.4.4 Konetestaus

Konetestausta tarvitaan, koska kaikkia ohjelmistovirheitä ei voida tai on hyvin hankala huomata simulaattorilla. Tyypillisesti koneelle asti pääsivät hyvin monimutkaiset tai johonkin kolmannen osapuolen komponenttiin liittyvät virheet. Konetestauksessa on myös oikeat massat, nopeudet ja muu fysiikka, jota simulaattorissa ei ole. Konetesteissä tehdään myös lopulliset parametrien säädöt.

3.4.5 Manuaalinen regressiotestaus

Toiminnallisessa testauksessa pyritään löytämään kehityksen aikana mahdollisimman paljon uusia virheitä ja virhetilanteita, regressiotestauksessa tarkistetaan, että ohjelmistoon tehdyt muutokset eivät ole rikkoneet jo toimivaksi todettuja ominaisuuksia. Samalla tarkistetaan, että jo löydetyt virheet ovat pysyneet korjattuina. (TestIO, [viitattu 23.2.2021].)

3.4.6 Automaattinen regressiotestaus

Regressiotestaus voidaan automatisoida koodaamalla testitapaukset ja käyttämällä testien ajamiseen siihen suunniteltua sovellusta. Regressiotestien automatisointi on kannattavaa, sillä testien ajaminen on paljon nopeampaa kuin manuaalisesti. Samalla myös testauksen kustannukset pienenevät. Automaattisen testaamisen etuna on myös se, että testaajan ei tarvitse ajaa manuaalisesti samoja testejä aina kun ohjelmistoon on tehty muutoksia. Testaajan aiheuttamia virheitä tulee vähemmän kun testit tehdään automaattisesti ja samalla tavalla joka kerta. Automaattisia testejä voidaan ajaa myös rinnakkain ja etänä. (ISTBQ, [viitattu 25.3.2021].)

3.5 Testauksen jako asiakkaan ja Epecin välillä

Yleisesti Epecin projekteissa asiakas vastaa konetestauksesta ja systeemi-integraatio-testauksesta, Epecin roolin ollessa avustava. Epec vastaa ohjausjärjestelmän yksikkö- ja integraatiotestauksesta käyttäen SIL- ja HIL-simulaattoreita. Jako voi kuitenkin vaihdella asiakkaittain.

3.6 Testauksen dokumentointi ja suunnittelu

Testaus sisältää tyypillisesti kolme vaihetta, jotka ovat testauksen suunnittelu, testaus sekä testien analysointi ja raportointi. Seuraavissa luvuissa syvennytään eri vaiheisiin.

3.6.1 Testauksen suunnittelu

Testaus suunnitellaan ALM-työkalussa esimerkiksi Polarionissa tai Jirassa. Testitapaukset olisi hyvä myös katselmoida jonkun muun kanssa ennen testausta. Näin voidaan välttyä virheelliseltä testaukselta. Näiden lisäksi testauksen suunnitteluun voi liittyä testausympäristön suunnittelua.

3.6.2 Testaaminen

Kun testitapaukset ovat valmiita ja tuote sekä testausympäristö ovat valmiina testaukseen, alkaa testien ajaminen. Testitapaukset ajetaan sovittua ohjelmaversiota vasten. Testitapauksien jokainen askel suoritetaan niin tarkasti kuin mahdollista ja kuitataan ne hyväksytyksi tai hylätyksi. Testitapaukset ajetaan manuaalisesti testaajan toimesta ellei automaattitestausta ole suunniteltu ja toteutettu projektiin.

3.6.3 Testien analysointi ja raportointi

Testien tulokset raportoidaan kun testit on ajettu. Raportti sisältää vähintään testitapauksien lopputuloksen. Se voi olla joko hyväksytty, epäonnistunut, estetty/ohitettu tai ei ajettavissa. Raportissa voi myös näkyä millä konfiguraatiolla se testit on tehty ja mitä ohjelmaversioita on käytetty.

3.7 Vikojen hallinta

Ihmiset tekevät virheitä, vaatimuksia päivitetään ja ympäristö voi muuttua niin, että vikoja tulee. Vikoja ei ole mahdollista poistaa kokonaan, mutta virheiden määrä ja erityisesti kriittisten vikojen määrä voidaan minimoida ennen järjestelmän julkaisemista. Tämän tavoitteen saavuttamiseksi on toteutettava virheenhallintaprosessi. Ohjelmistovirheet ovat kehittäjälle kalliita, ylläpito vaiheessa havaittu vika voi maksaa joka 40–1000 kertaisesti (Boehm 1981, 40.) kuten taulukosta 1 voidaan havaita:

Taulukko 1 Vikojen korjaamisen kustannuskertoimet (Boehm 1981, 40.)

Vika löydetty vaiheessa	Kustannuskerroin
Vaatimusten määrittely	1
Suunnittelu	3-5
Kehitys	10
Moduuli testaus	15-40
Funktionaalinen testaus	30-70
Systeemi testaus	40-90
Tuotanto	40-1000

3.7.1 Vian elinkaari

Kun ohjelmistovirhe havaitaan, sen on seurattava hallittua elinkaarta. Muussa tapauksessa tapauksesta ei ehkä raportoida, huomata ja korjata. Epecin projekteissa vian elinkaariprosessissa on neljä päävaihetta sen jälkeen, kun uusi vika on löydetty. Päävaiheet ovat raportointi, validointi, päätös ja verifiointi.

Kun uusi vika löydetään, se ilmoitetaan uutena vikana asianmukaisilla tiedoilla. Näihin tietoihin on sisällyttävä vian tueksi saadut tiedot, jotta ne voidaan analysoida, onko tapaus pätevä vai ei. Perustiedot, jotka on liitettävä ohjelmointivirheeseen, ovat otsikko, kuvaus, ohjelmistoversio, vaiheet ongelman toistamiseen, vakavuus ja linkitykset vastaaviin toimintoihin, esimerkiksi ominaisuuksiin, vaatimuksiin ja testitapauksiin.

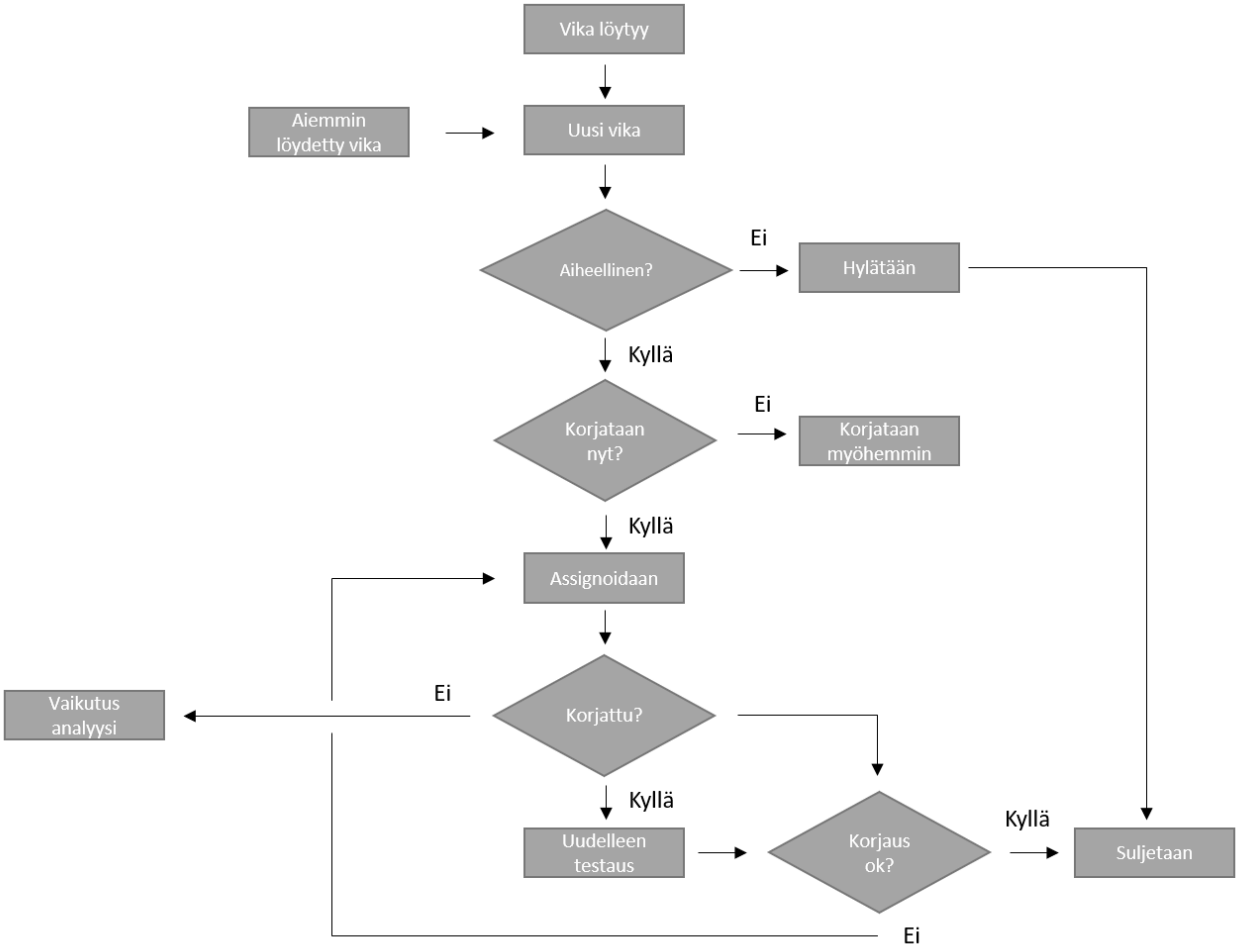
Vian ilmoittamisen jälkeen on päätettävä, onko havaittu vika validi vai ei. Jos päätetään, että vika on validi, se osoitetaan kehittäjälle lisätutkimuksia varten. Tutkinnan yhteydessä on analysoitava ongelman lähde, että voidaan selvittää, onko virheellä vaikutusta esimerkiksi vaatimuksiin tai onko sillä vaikutusta muuallekin tuotteeseen. Jos taas päätetään, että vika ei ole validi, se hylätään ja suljetaan.

Kun päätetään, että vika on validi, on ryhdyttävä korjaaviin toimenpiteisiin. Korjaava toimenpide voi olla se, että ei tehdä mitään juuri nyt, jolloin korjausta lykätään ja se avataan uudelleen uutena vikana ja analysoidaan uudelleen, jos tämä nähdään projektin myöhemmissä vaiheissa tarpeelliseksi. Toinen korjaava toimenpide on se, että muutokset on tehtävä välittömästi, jolloin vika on korjattava ja korjaus varmistettava testaamalla.

Kun kehittäjä on tehnyt korjauksen, hän osoittaa vian testaajalle ja asettaa vian tilaksi ”valmis testattavaksi”. Jos kehittäjä jostain syystä havaitsee tutkimuksen aikana, että ongelmaa ei voida korjata tai sitä ei tarvitse korjata, vian tilaksi asetetaan ”hylätty”. Hylätyt viat suljetaan, mikäli todetaan, että vikaa ei tarvitse korjata.

Kun testaaja on verifioinut korjauksen, vian tilaksi asetetaan ”korjattu”. Mikäli korjaus ei poista virheellistä toimintaa, vika osoitetaan kehittäjälle uudelleen ja tilaksi asetetaan ”avoin”. Testaajan on analysoitava ongelma varmistaakseen, että tämänhetkinen testaustapaus kattaa ongelman.

Kuviossa 3 kuvataan vian elinkaaren prosessi.



Kuvio 3. Vian elinkaaren prosessi

3.7.2 Vikojen ilmoittaminen ja seuranta

Viat raportoidaan projekteissa testaussuunnitelmassa määritellyllä tavalla, mukaillen luvussa 3.7.1. määriteltyjä tapoja. Kaikki viat ilmoitetaan ja tallennetaan sovitulla tavalla käytössä olevaan ALM-työkaluun.

3.7.3 Vikojen vakavuuden määritelmät

Vian vakavuuden määritelmiä käytetään ilmaisemaan virheen aiheuttaman ohjelmiston laatuun kohdistuva negatiivinen vaikutus. Yleisesti Epecin projekteissa käytetään seuraavia luokituksia:

Taulukko 2. Vian vakavuuden luokittelu

(S1): Kriittinen	Vika vaikuttaa kriittiseen toiminnallisuuteen, eikä sitä voi ohittaa. Esimerkki kriittisestä virheestä: • Kriittisen toiminnallisen ominaisuuden täydellinen epäonnistuminen
(S2): Korkea	Vika vaikuttaa ei-kriittiseen toiminnallisuuteen, eikä sitä voi ohittaa. Esimerkki korkeasta virheestä: • Ei-kriittinen toiminnallinen ominaisuus toimii osittain
(S3): Keskiverto	Vika vaikuttaa ei-kriittiseen toiminnallisuuteen, eikä sitä tarvitse ohittaa. Esimerkki keskivertoviasta: • Ei-kriittinen toiminnallinen ominaisuus toimii osittain
(S4): Matala	Vika on kosmeettinen (esim. käyttöliittymään liittyvät ongelmat, ulkoasu tai oikeinkirjoitus ei ole oikea)

3.8 Laadunhallinta koodauksen ja testauksen aikana

Laadunhallinta on kahden lähestymistavan yhdistelmä: laadunvarmistuksen ja laadunvalvonnan. Laadunvarmistus on staattista testausta, jonka ensisijainen tarkoitus on estää vikoja. Laadunvarmistusta toteutetaan koko tuotteen elinkaaren ajan vaatimusvaiheesta projektin sulkemiseen. Laadunvarmistukseen kuuluvat erilaiset katselmoinnit ja läpikäynnit. Laadunvalvonta taas on dynaamista testausta, jonka ensisijaisena tarkoituksena on löytää vikoja. Laadunvalvonnassa testataan tuotetta, jotta voidaan varmistaa, että se toimii määriteltyjen vaatimusten mukaisesti. (ToolsQA 20.9.2019.) Laadunvalvontaan kuuluvat luvun 3.4 eri testausvaiheet.

3.8.1 Vaatimuksien tulkinnanvaraisuus

Vaatimuksien tulisi olla yksiselitteisiä ettei niihin jää tulkinnan varaa. Vaatimuksien tulee aina olla niin selkeitä, että niiden perusteella koodaaja osaa tehdä halutunlaisen ohjelman ja testaaja osaa helposti luoda testit sekä testata ne. Kun vaatimukset on hyvin ja selkeästi tehty, ei väärinkäsityksiä pääse syntymään. (Stenberg 2020.)

3.8.2 Katselmointikäytännöt

Katselmointikäytäntöjä on erilaisia ja niiden käyttäminen riippuu projektin kriittisyydestä ja siitä, mitä katselmoidaan. Katselmointeja voidaan pitää muun muassa vaatimusmäärittelyille, koodeille ja testisuunnitelmille sekä testitapauksille. Katselmointien päämääränä on löytää viat ja puutteet. Katselmoinnit on hyvin tärkeä osa laadunvarmistusta, koska testaus yksistään ei kata läheskään kaikkea. Katselmoinnit ovat lisäksi halvempia jopa samojen virheiden löytämiseen kuin testaus (Basili & Selby 1987, 733.) Katselmointien muita hyötyjä ovat, että siihen osallistujat voivat oppia ja ymmärtää ohjelmaa paremmin, voidaan ennaltaehkäistä virheitä ja projektin läpimenoaika lyhenee (Gilb 2000, 46).

3.8.3 Laadunvarmistuksen onnistuminen

Laadunvarmistus on onnistunut silloin kun virheet huomataan hyvin aikaisessa vaiheessa ja tuote vastaa asiakkaan toiveita. Tässä pitää myös huomioida minkälaisella panostuksella kyseiseen lopputulokseen päästiin. Mitä huonommin laadunvarmistus on onnistunut, sitä myöhemmässä vaiheessa ohjelmointivirheet huomataan ja ohjelmaan joudutaan tekemään muutoksia, sillä se ei vastaa asiakkaan toiveita.

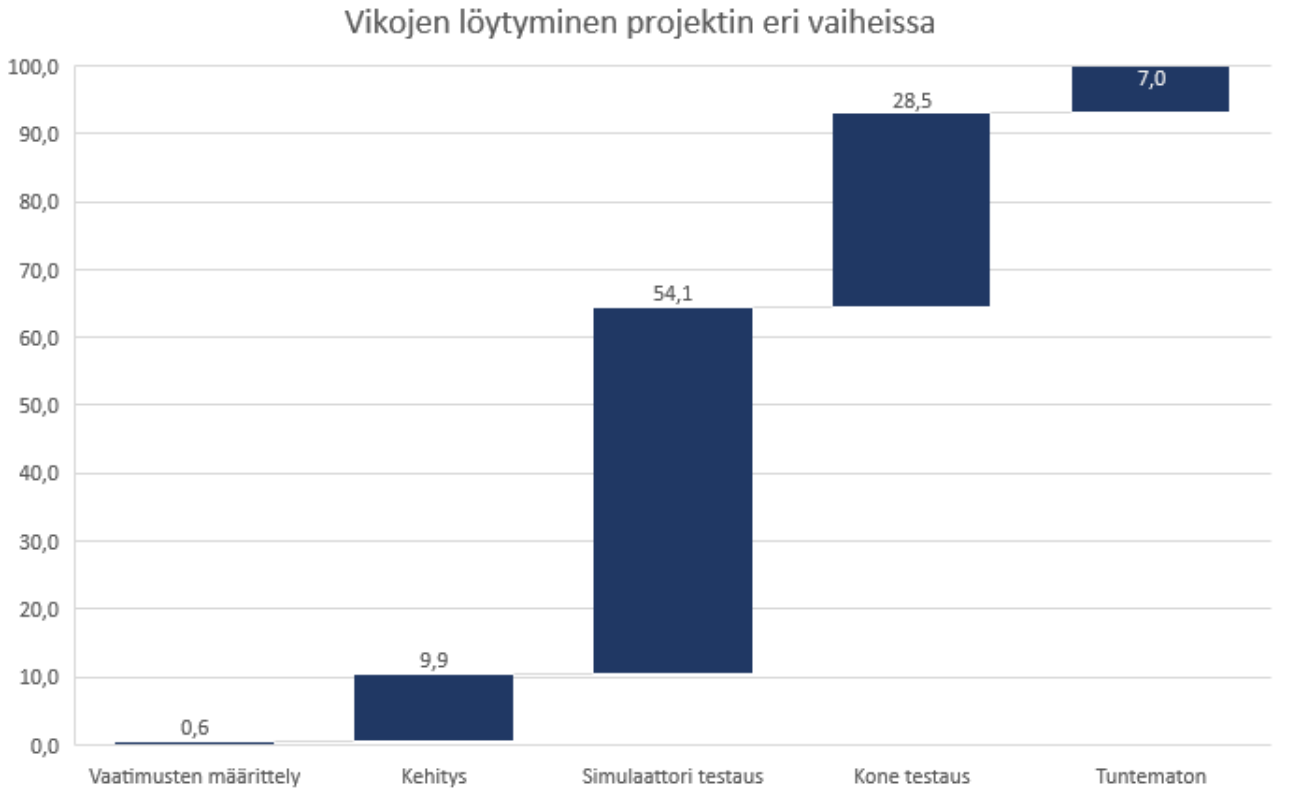
3.9 Analyysi asiakasprojektin laadunvarmistuksen onnistumisesta

Tässä luvussa käsitellään ja analysoidaan Epecin erään asiakasprojektin laadunvarmistuksesta saatua dataa, jonka pohjalta tehdään päätelmiä laadunvarmistuksen onnistumisesta kyseissä projektissa.

Projektissa oli käytössä ALM-työkaluna Polarion, jossa oli koneen vaatimukset, spesifioinnit, testisuunnitelmat, testitapaukset ja viat sekä niiden hallinta. Vian ilmetessä vika-työkortin ehtoihin oli määritelty, että siihen tulee kirjata myös missä vaiheessa vika löytyi ja missä vaiheessa sen olisi pitänyt löytyä. Kirjauksen teki omaa harkintaa käyttäen vian löytäjä.

3.9.1 Vikojen löytyminen

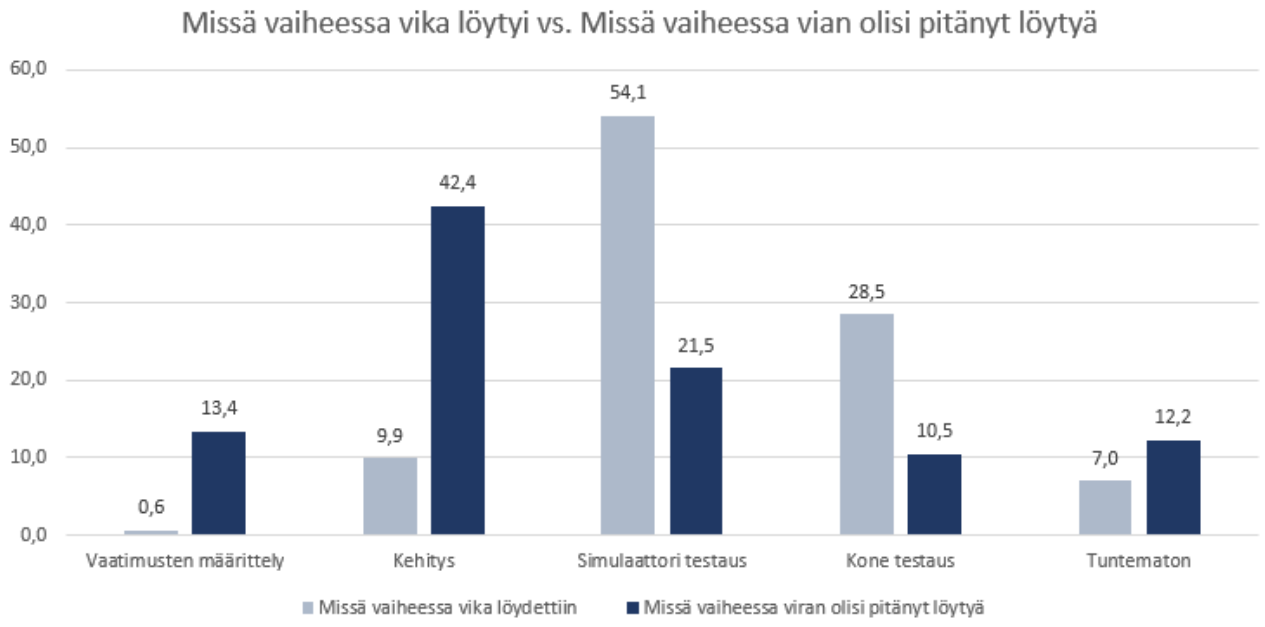
Työn esimerkkiprojektissa vikojen löytäminen projektin eri vaiheissa jakaantui kuviossa 4 esitetyllä tavalla.



Kuvio 4 Vikojen löytyminen projektin eri vaiheissa

Selvästikin merkittävin osa vioista löytyi jo simulaattoritesteissä.

Kun tarkasteltiin, missä vaiheessa vikoja löytyi verrattuna siihen, missä vaiheessa ne olisi pitänyt löytää, merkittävin osa olisi pitänyt huomata jo kehityksessä sekä jo vaatimuksia määritellessä.



Kuvio 5 Missä vaiheessa vika löytyi vs. Missä vaiheessa vian olisi pitänyt löytyä

Tuloksista voidaan päätellä, että virtuaalisimulaattorilla saadaan suuri osa vioista kiinni ennen kuin ne pääsevät koneelle asti. Kuitenkin voidaan todeta, että kaikkia vikoja ei voida tai on hyvin hankala huomata simulaattorilla. Tyypillisesti koneelle asti pääsivät hyvin monimutkaiset tai johonkin kolmannen osapuolen komponenttiin liittyvät viat.

Kuten luvussa 3.7 todetaan viat olisi tärkeä löytää jo hyvin varhaisessa vaiheessa, koska niiden kustannukset kasvavat moninkertaisiksi mitä myöhemmin ne havaitaan. Vikojen mukana oli myös sellaisia, joista puuttui tieto, missä vaiheessa ne löytyivät tai missä vaiheessa ne olisi pitänyt löytyä. Vian kirjauksessa tulisi olla huolellisempi, että kaikki tiedot kirjataan ylös.

3.9.2 Poikkeavat ja muista erottuvat viat

Tässä luvussa käsitellään muutamia vikoja, jotka nousivat esille muiden vikojen joukosta haastavuudellaan, erikoisuudellaan tai sillä että niiden korjaaminen kesti kauan.

Ensimmäinen käsiteltävä vika oli virtuaalisimulaattorilla löytynyt vika toiminnosta, jota ohjataan joystick-ohjaimella ja sen liikettä vaimennetaan parametreilla annettujen arvojen mukaan. Alasohjatessa vaimennus ei toiminut, mutta ylösohjatessa se toimi. Vikaa toistettiin testaajan toimesta niin, että koodaaja tarkasteli koodia, kun vika toistettiin. Tällöin oli tehty virheellinen tulkinta, ettei vaimennus toimi simulaattorissa, koska joystickille tuli negatiivinen arvo alasohjattaessa ja päätettiin, että testaus jätetään konetesteihin. Vika kuitenkin toistui myös konetesteissä. Vikaa alettiin tutkia uudelleen, jolloin selvitettiin, että muuttuvatko parametrit IO-koodissa, kun niitä säätää näytöltä. Vasta tässä vaiheessa havaittiin, että lähes kaksi saman nimistä ja peräkkäistä parametria olivat menneet väärinpäin. Pieni vika aiheutti kauan auki olevan vian ja siihen käytettiin yli 20 h aikaa pelkästään Epecin toimesta. Automatisoimalla Python-skriptillä parametrien generoinnit koodiin, tälläisiltä vioilta voitaisiin välttyä.

Toinen käsiteltävä vika löytyi myös virtuaalisimulaattorilla, sama vika huomattiin myös koneella jälkeenpäin. Vika löytyi toiminnosta, jossa pakotetaan näytön ikkunan kautta koneen toimintoja. Ikkunassa 7/10 pakotuksista toimi, mutta loput kolme eivät toimineet. Heti huomattiin, että yhdeltä ei lähtenyt signaalia näytöltä ja kahden pakotuksilta puuttuivat toteutukset kokonaan IO-koodista. Niiden korjaamisen jälkeen vika palautui testaajalle, mutta ne eivät toimineet vieläkään ja vika palautui korjattavaksi. Koodaaja havaitsi tässä vaiheessa, ettei näytön koodissa kaikkien signaalien lähetys ollut vieläkään kunnossa, lisäksi testeissä huomattiin, että yksi lähetettävä arvo tuli IO-koodiin väärässä muodossa. Tämän vian korjausprosessi kesti yli 5 kk.

Kolmas vika johtui itse virtuaalisimulaattorista, eli se ei ollut aiheellinen vika. Toiminnon sekvenssiä testatessa huomattiin, että 120 s kestävä sekvenssiaika kestääkin simulaattoritesteissä 240 s. Sykliaika ei ole oikea virtuaalisimulaattorissa, jolloin syklipohjaiset laskurit toimivat hitaammin kuin todellisuudessa.

Lisäksi esille nousi vikoja, jotka olisi pitänyt löytää simulaattori testeissä, mutta ne löytyivät vasta koneella. Yleensä nämä olivat sellaisia, jotka eivät vaikuttaneet simulaatiomallin toimivuuteen tai koneen käytettävyyteen. Esimerkiksi jännitteiden, virtojen, tehojen ja paineiden oikeellisuus oli hankala huomata simulaattorissa ellei niitä oltu määritelty koneen vaatimuksissa selkeästi. Myös ajonopeuksiin ja ajosuuntaan liittyvät toiminnallisuudet oli haastava testata, kunnes simulaattoriin tehtiin Python-skripti, joka laski nopeuden oikealla tavalla. Muutamia vikoja oli myös puutteellisten tai virheellisten vaatimuksien takia, lisäksi osa vaatimuksista muuttui kehityksen aikana.

4 Testauksen kehitys

4.1 Miten tulevaisuus tulee vaikuttamaan testaukseen

Tulevaisuudessa ohjausjärjestelmät tulee yhä enemmän monimutkaistumaan. Tällä hetkellä ja tulevaisuudessa koneet sähköistyvät ja avustavia järjestelmiä tulee koneisiin. Lisäksi autonomisia koneita alkaa tulla kasvavissa määrin. (Stenberg, 2020.) Nämä kaikki vaikuttavat myös testauksen ja laadunvarmistuksen monimutkaistumiseen sekä vaativuuteen. Samalla myös testausohjelmistot sekä simulaattorit kehittyvät ja antavat parempia keinoja ohjelmistojen ja koneiden testaamiseen.

4.2 Analyysin perusteella löydetyt asiat, joissa on kehitettävää tai pitäisi toimia toisin

Luvun 3.9 analyysin perusteella voidaan havaita, että vaatimusten määrittelyvaiheen ja kehityksen aikaista katselmointia pitäisi kehittää. Projektia suunnitellessa täytyy määritellä tarkemmin, mitä ja miten katselmoidaan ja seurata niiden toteutumista. Katselmointikäytännöt olisi hyvä ottaa myös pienemmissä projekteissa käyttöön. Katselmointiin olisi syytä tutkia myös erilaisia työkaluja ja tutkia, onko esimerkiksi jo käytössä olevissa ALM-työkaluissa helpottavia ominaisuuksia, joita voidaan hyödyntää.

Virheellisten tai väärin ymmärrettyjen vaatimusten ehkäisemiseksi pitäisi myös panostaa Epecin, asiakkaan ja kolmannen osapuolen kommunikointiin ja yhteispalavereihin. Tällöin voitaisiin mahdollisesti välttyä tilanteilta, joissa tiedon kulku kestää kauan ja viivyttää kehityksen etenemistä, tai jokin ei toimikkaan kuten asiakas on määritellyt. Myös spesifiointivaiheessa olisi hyvä saada tietoa henkilöiltä, jotka konetta tulevat testaamaan tai käyttämään, jolloin saisi käytön kannalta oleellisia asioita jo siinä vaiheessa paremmiksi.

Voidaan myös havaita, että virtuaalisimulaattorilla saadaan suuri osa vioista kiinni. Niihin tulisi jatkossa käyttää enemmän resursseja sekä tutkia voidaanko virtuaalisimulaattoreita parantaa tai onko parempia vaihtoehtoja. Aina, jos projekti on riittävän laaja, olisi hyvä hankkia sen testaamiseen virtuaalisimulaattori. Simulaattorien käyttö vaatii myös tietokoneelta paljon tehoa verrattuna yksinkertaisen VTE-käyttöliittymän käyttöön, joten sekin tulee huomioida.

5 Pohdintaa ja yhteenveto

Opinnäytetyön tavoitteena oli selvittää testauksen nykytilaa Epecillä ja mahdollisia parantamiskohteita. Esimerkkiasiakasprojektin tuloksia analysoimalla selvisikin, että kehitettävää on katselmointikäytännöissä, virtuaalisimulaattoreihin pitäisi panostaa ja tarvittaessa kommunikointia lisätä yrityksen, asiakkaan ja kolmannen osapuolen välillä. Tarkoituksena oli myös kerätä tietoa, jota voidaan hyödyntää tulevien projektien testauksen suunnittelussa.

Testausmenetelmistä ja vaiheista kirjoittaessa havaittiin, että ohjausjärjestelmien testaukseen ei suoranaisesti ole kovin kattavasti kirjallisuutta tai tietoa saatavilla, mutta ohjelmistotestaukseen yleisesti kirjoitetuista teoksista ja verkkojulkaisuista sai työhön hyvin tietoa. Myös käytetyt termit ja niiden tarkoitukset vaihtelivat jonkin verran, joten olisikin syytä selventää näitä yrityksen sisällä, jotta kaikille on selvää, mistä milloinkin on kyse.

LÄHTEET

About Polarion Software n.d. Ei päiväystä. Polarion ALM. [Verkkosivu]. Siemens Industry Software Inc. [Viitattu 12.11.2020]. Saatavilla: https://polarion.plm.automation.siemens.com/products/polarion-alm

Azure Test Plans Microsoft. Ei päiväystä. [Verkkosivu]. Microsoft Corp. [Viitattu 12.11.2020] Saatavilla: https://azure.microsoft.com/en-us/services/devops/test-plans/

Basili, V.R. & Selby, R. 1987. Comparing the effectiveness of software testing strategies. IEEE Transactions on Software Engineering 13(12) 1278-1296.

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

Bourque, P. & Fairley, R.E. 2014. Guide to the Software Engineering Body of Knowledge (SWEBOK). IEEE Computer Society

Bringmann, E. & Krämer, A. 2008. Model-based Testing of Automotive Systems. [Verkkojulkaisu]. PikeTec GmbH. [Viitattu 26.2.2021]. Saatavilla: https://www.win.tue.nl/~mvdbrand/courses/sse/0809/papers/MBT.pdf

Epec. Ei päiväystä. Epec company presentation 2020. [Verkkojulkaisu]. Epec Oy. [Viitattu 15.2.2021]. Saatavilla: Vain yrityksen sisäisessä käytössä.

Epec. Ei päiväystä. Epec VTE koulutus. [Verkkojulkaisu]. Epec Oy. [Viitattu 10.11.2020]. Saatavilla: Vain yrityksen sisäisessä käytössä.

Gartner. 29.8.2019. [Verkkosivu]. Gartner. [Viitattu 15.2.2020] Saatavana: https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io

Gilb T. 2000. Planning to get the most out of inspections. Software Quality Professional, vol 2, no 2, 2000.

Guru99. Ei päiväystä. Explorary Testing. [Verkkosivu]. Guru99. [Viitattu 20.11.2020]. Saatavilla: https://www.guru99.com/exploratory-testing.html

Guru99. Ei päiväystä. Integration Testing. [Verkkosivu]. Guru99. [Viitattu 15.2.2021]. Saatavilla: https://www.guru99.com/integration-testing.html

Guru99. Ei päiväystä. System Integration Testing. [Verkkosivu]. Guru99. [Viitattu 15.2.2021]. Saatavilla: https://www.guru99.com/system-integration-testing.html

ISTBQ. Ei päiväystä. Advanced-Test-Automation-Engineer-Syllabus-GA-2016 [Verkkojulkaisu]. ISTBQ. [Viitattu 25.3.2021]. Saatavilla: https://www.istqb.org/certification-path-root/test-automation-engineer.html#syllabus

Jira. Ei päiväystä. Jira Features [Verkkosivu]. Atlassian. [Viitattu 12.11.2020]. Saatavilla: https://www.atlassian.com/software/jira/features

Myers, G.J., Sandler, C. & Badgett, T. 2012, The art of software testing, John Wiley & Sons, Hoboken, N.J.

Stenberg, A.Senior QA Consultant. Q-Factory Oy. Teams-haastattelu 12.10.2020.

TestIO. Ei päiväystä. Regression testing. [Verkkosivu]. TestIO. [Viitattu 23.2.2021]. Saatavilla: https://test.io/regression-testing/

ToolsQA. 20.9.2019. Quality Assurance and Quality Control. [Verkkosivu]. ToolsQA. [Viitattu 10.2.2021]. Saatavilla: https://www.toolsqa.com/software-testing/istqb/quality-assurance-and-quality-control/