Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Mikko Määttänen

Ohjelmistotestauksen automatisointi

Metropolia Ammattikorkeakoulu

YAMK

Älykäs teollisuus

Ohjelmistotestauksen automatisointi

27.4.2021

Tekijä(t) Otsikko Sivumäärä Aika	Mikko Määttänen Ohjelmistotestauksen automatisointi 43 sivua 27.4.2021
Tutkinto	YAMK
Tutkinto-ohjelma	Älykäs teollisuus
Ohjaaja(t)	Tutkintovastaava Jarno Varteva Lehtori Tuomo Heikkinen

Opinnäytetyön tavoitteena oli selvittää testauksen automatisoinnin ottamista osaksi kohdeyrityksen Mipro Oy ohjelmistokehityksen prosessia. Työssä tarkasteltiin ohjelmistotestauksen automatisoinnin koko prosessia: tutkimuksen lähtökohtia, yrityksen prosessimallia, standardeja, vaatimuksia, testausta kokonaisuutena ja automatisointia. Tutkimus pyrki antamaan lukijalle yhdenlaisen ratkaisun kuinka ohjelmistotestauksen automatisointi voidaan toteuttaa PK-yrityksessä.

Aluksi selvitettiin automatisoinnin tarpeellisuutta ja tavoitteita yleisesti sekä erityisesti Mipro Oy:ssä. Tämän jälkeen esiteltiin Mipro Oy ja kuvataan ohjelmisto, jonka testaamiseen automatisointia tultiin käyttämään. Käytiin läpi yrityksen toimintaprosessia sekä standardeja ja vaatimuksia.

Esiteltiin kehitysvaiheen yleisimmät testausmenetelmät ja testauksen eri tasot. Kuvailtiin tärkeimmät asiat testauksen olennaisimmista dokumenteista: lähtötiedoista, testaussuunnitelmasta, vikaraportista sekä testausraportista.

Työn lopussa tavoitteena oli esitellä kuinka automatisoinnin tuloksia mittarointiin. Tämä prosessin vaihe ei kuitenkaan ennättänyt valmistumaan työn kirjoittamisen päättyessä. Työssä selvitettiin kuitenkin mittarointia teoriatasolla ja siihen liittyvää suunnittelua, vaatimuksia, sekä hyötyjä ja haittoja.

Avainsanat	Testaus, automatisointi, Robot Framework

Author(s) Title Number of Pages Date	Mikko Määttänen Automating Software Testing 43 pages 27 Apr 2021
Degree	Master Degree Program
Degree Programme	Intelligent Industry Research Program
Instructor(s)	Jarno Varteva, Degree Equivalent Tuomo Heikkinen, Senior Lecturer

The aim of the thesis was to find out how testing automation would be included in the software development process in the target company Mipro Oy. The work examined the whole process of software testing automation: research starting points, company process model, standards, requirements, development phase testing methods, different levels of testing, testing documentation, automation and its implementation in the company and finally measurement of results. The research aimed to give the reader one kind of a solution on how software automation can be implemented in a small to medium-sized company.

Initially, the need for and goals of automation in general and specifically in Mipro Oy were investigated. After this Mipro Oy was introduced and the software for which automation was to be used was described. The company's business processes, standards and requirements were reviewed. This formed the framework for the entire software development.

The most common testing methods and different levels of testing during the development phase were presented. The most important elements of the most essential testing docum- were described: the initial data, the test plan, the fault report and the test report.

The thesis also looked at how the new testing method was implemented in the organization and shared the findings on the advantages and disadvantages relating to the implementation of the new testing methdod.

At the end of the thesis objective was to present how the results of the automation are measured. However, this stage of the process was not quite get completed by the end of finishing the thesis work due to factors beyond the writer’s control. Requirements, planning, advantages and disadvantages of metering at the theoretical level were investigated.

Keywords	Testaus, automatisointi, Robot Framework

Sisällys

1 Tutkimuksen lähtökohta 4

1.1 Automatisoinnin tarpeellisuus ja tavoitteet 4
1.2 Kohdeorganisaation esittely 5
1.3 Testattavan tuotteen kuvaus 6
1.3.1 Opastimet 7
1.3.2 Vaihde ja vaihteen kääntölaitteet 12
1.3.3 Raiteen sulku ja pysäytyslaite 15
1.3.4 Varmistuslukko ja avainsalpalaite 16

2 Prosessimalli, standardit ja vaatimukset 17

2.1 Standardit ja ohjeistukset 17

3 Kehitysvaiheen testausmenetelmät 21

3.1 Black box – testaus 21
3.2 White box – testaus 21
3.3 Gray box – testaus 22
3.4 Regressiotestaus 22
3.5 Kuormitustestaus 23

4 Testauksen tasot 24

4.1 Yksikkötestaus 24
4.2 Integrointitestaus 24
4.3 Järjestelmätestaus 25
4.4 HW testaus 26
4.5 Hyväksymistestaus 26
4.6 Käyttöönottotestaus 26

5 Testauksen dokumentaatio 26

5.1 Lähtötiedot 27
5.2 Testaussuunnitelma 27
5.2.1 Testauspolitiikka 27
5.2.2 Testausstrategia 28
5.3 Testitapaukset 29
5.4 Vikaraportointi 30
5.5 Testausraportti 31

6 Automatisoinnin toteutus 32

6.1 Yleistä 32
6.2 Robot Framework 32
6.2.1 Yleistä 32
6.2.2 Esimerkki 33
7 Automatisoinnin jalkauttaminen organisaatioon 37
7.1 Käyttöönotossa huomioitavat yleiset asiat 37
7.1.1 Työvälineet ja testausympäristö 38
7.1.2 Koulutukset 38
7.1.3 Resurssointi 38
7.2 Käyttöönotto 38
7.3 Käyttöönoton välittömät hyödyt ja haitat 39
8 Tulosten mittarointi 39
8.1 Yleistä mittaroinnista 39
8.2 Mittaroinnin vaatimukset 40
8.3 Mittariston suunnittelu 40
8.4 Mittaroinnin hyödyt ja haitat 41
9 Yhteenveto ja pohdintaa 42
Lähteet 43

Lyhenteet

PK – yritys Pieni Keskisuuri yritys. Yritys, joka työllistää alle 250 henkilöä.

DevOps Development Operations. On toimintamalli pyrkii automatisoimaan ohjelmistokehitykseen, testaamiseen ja ylläpitoon liittyvät IT-palvelutoiminnot.

HW Hardware

SIL Safety Integrity Level eli turvallisuuden eheyden taso. IEC:n (International Electrotechnical Commision) käyttämä turvavaatimustaso.

Verifiointi Verifiointi eli todentaminen eli tarkastus. Tarkastetaan siis, että järjestelmä on tehty oikein ja menettelyjä on käytetty oikein.

Validointi Validointi eli kelpuutus tai hyväksyntä. Sen tarkoituksena on myös varmistaa, että järjestelmä on vaatimusten mukainen ja tehty oikeilla menettelyillä.

Beta testaus

Ohjelmiston beta -versiot julkaistaan rajatulle yleisölle, ohjelmistokehitysryhmän ulkopuolelle koekäyttöön.

Olio Olio-ohjelmoinnissa ohjelmiston perusyksikkö, joka sisältää joukon loogisesti yhteenkuuluvaa tietoa ja toiminnallisuutta.

Tynkä Englannin kielessä käytetty *Stub* tarkoittaa niin sanottua sijaiskomponenttia mitä käytetään yleensä testaamiseen silloin kun varsinainen komponentti ei ole vielä valmis.

TFFR Tolerable Functional (unsafe) Failure Rate, hyväksytyn toimintahäiriön raja. Määritteleen hyväksytyn toimintahäiriöiden raja-arvot.

Suojastus Toimintojen muodostama kokonaisuus, jolla varmistetaan asetinlaitteen varmistamaa kulkutietä vastaava reitti linjalla.

Suojastusosuus

Suojastuksen varmistama kulkutietä vastaava reitti linjalla.

EMC Electromagnetic compatibility, sähkömagneettinen yhteensopivuus.

RAMS Reliability Availability Maintainability and Safety (luotettavuus, saatavuus, ylläpidettävyys ja turvallisuus).

ATDD Acceptance test-driven development, hyväksymistestausvetoinen ohjelmistokehitys

Python Ohjelmointikieli

RPA Robotic Process Automation, robottiprosessiautomaatio

JVM Jython, Pythonin Java-kielinen toteutus

IronPython Pythonin C#-kielinen toteutus

Abstraktointi

Tarkoittaa teoreettisten käsitteiden luomista. Tässä yhteydessä tyyppien ymmärrettävä nimeäminen.

Product Owner

Scrum tiimin jäsen, joka vastaa tuotteen arvon ja kehitystiimin työn arvon maksimoimisesta.

Sprintti Scrum työskentelyssä olevan kehitysjakso, jonka pituus voi vaihdella 1-4 vikkoa.

Scrum Ketterä työskentelymenetelmä, joka muodostuu sprinteistä.

Johdanto

Ohjelmistokokonaisuuksien jatkuvasti kasvaessa ja monimutkaistuessa vaaditaan niiden testaamiseen yhä enemmän voimavaroja. Testauksen automatisointi on nähty yhtenä merkittävänä apuna muodostuneeseen tilanteeseen. Monelle automatisointi voi merkitä lähes kaiken testaamisen automatisointia. Sitä se kuitenkin hyvin harvoissa tilanteissa merkitsee. Haasteita automatisointiin voivat tuoda oikeanlaisten työkalujen valitseminen ja uudenlaisen osaamisen löytäminen. Automatisoinnin tarpeellisuuden arviointi ja suunnitelmallinen toteutus ovat avainasemassa oikeanlaisen testauksen löytymiseen.

Työn tavoitteena on selvittää miten ohjelmistotestauksen automatisoinnin toteutus kohdeyrityksessä Mipro Oy:ssä hoidettiin. Selvitetään miksi ohjelmistotestausta on tarve automatisoida? Mitä hyötyjä ja haittoja automatisoinnista syntyy? Pohditaan mitä kaikkea toimivien mittarien suunnittelussa ja toteutuksessa tulisi huomioida ja miten mittarointi suunnitellaan.

Tavoitteena on esitellä testausta myös kokonaisuutena. Mistä kaikista osista testaus koostuu? Esitellään testauksen peruspilarit. Lopussa selvitetään miten automatisoinnin jalkauttaminen tulisi toteuttaa.

Työn aihe valittiin kirjoittajan toivomuksesta yhteispäätöksellä Mipro Oy:n kanssa. Automatisointiprosessi oli opinnäytetyön aihetta valittaessa jo käynnistynyt joten työhön liittyvää aineistoa oli mahdollista päästä heti tutkimaan ja dokumentoimaan. Miprolle työn merkitys on lähinnä se, että työ luo kokonaiskuvauksen automatisointiprosessin toteuttamisesta unohtamatta testauksen perusasioita, joiden päälle myös testausautomaatio rakentuu.

1 Tutkimuksen lähtökohta

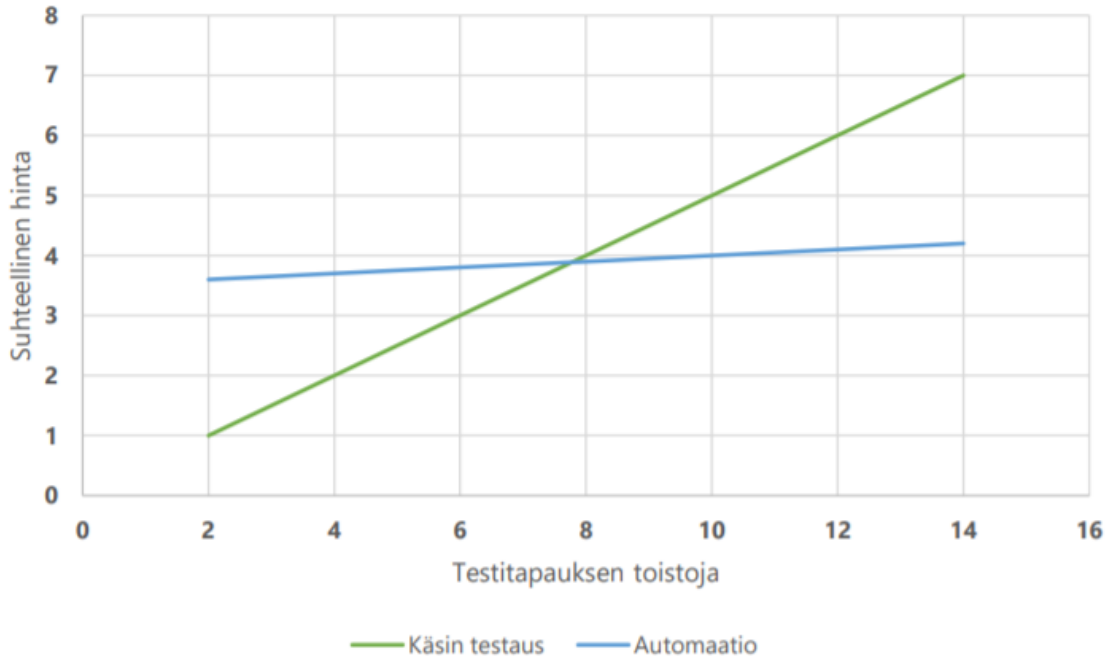
1.1 Automatisoinnin tarpeellisuus ja tavoitteet

Jatkuvan julkaisu toimintamalliin kuuluu olennaisena osana testausautomaatio. Katkeamattoman toimituksen edellytyksenä on, että ohjelmisto testataan välittömästi uuden version valmistumisen jälkeen. Ihminen tekee sen jälkeen vielä päätöksen julkaistaanko uusi versio vai ei. Jatkuvaan julkaisuun on siirrytty digitalisoitumisen ja pilvipalveluiden yleistyttyä.

Jatkuvan julkaisun edellytyksenä on, että yrityksen prosessit ja myös kulttuuri vastaavat jatkuvan julkaisun edellyttämiä vaatimuksia. Kehitystiimin työn on oltava ketterää DevOps –tyyppistä ja sen on pyrittävä integroimaan jatkuvan julkaisun muiden töiden yhteyteen (Banerjee K. 2018).

Testausautomaatiota käytetään testauksen eri vaiheissa kuten; yksikkö-, integrointi-, järjestelmä-, regressio- sekä suorituskykytestauksessa. Sen päätarkoituksena on vähentää manuaalisen testaamisen tarvetta, parantaa tuoteen laatua, vähentää kustannuksia ja nopeuttaa testausprosessia. Käsin testauksen merkittävästi vähentyessä, tuotteen kehitysprosessi nopeutuu (Kasurinen 2013, s. 76.). Isoilla digitaalisia palveluja tarjoavilla yrityksillä kuten Twitter, Facebook ja Google testausautomaation merkitys tuotekehityksessä on kriittinen. Palveluissa tehdään useita päivityksiä tiheällä syklillä (Feitelson, Frachtenberg & Beck 2013, s. 8).

Testausautomaatio vähentää ohjelmistokehityksen kustannuksia testien suorituskertojen kasvaessa. Usein testausautomaatiota ei lähdetä tekemään, koska automaation käynnistämisvaiheeseen liittyy paljon kustannuksia kuten; resurssit, evaluointiprosessi, mahdolliset lisenssit, valitun menetelmän sertifiointi, laitteisto ja ehkä suurimpana henkilöstön koulutus. Käsin testaaminen on siis vähäisillä toistokerroilla edullisempaa. Automaatiota ei siksi kannata lähteä perustamaan pienissä projekteissa, joissa sen käyttö jäisi vähäiseksi (Kasurinen 2010, s. 14). Kuvasta 1 selviää miten taloudellisesta näkökulmasta automaattinen testaaminen muuttuu kannattavaksi noin kahdeksannen testitoistokerran jälkeen (Kasurinen 2013, s. 78).



Kuva 1. Testauksen automatisoinnin kannattavuus suhteessa hintaan ja testien toistokertoihin (Kasurinen 2013, s. 78)

Testausautomaation ansiosta resursseja vapautuu tutkivan testaamisen tekemiseen mikä osaltaan täydentää tuotteen laadun monipuolisempaa tarkastelua (Kasurinen 2013, s. 78). Automatisointi tuottaa varsinaisen testauksen lisäksi myös muita testausta hyödyntäviä asioita kuten; virheiden monitoroinnin, tulosten nopean analysoinnin sekä testauksen raporttien automaattisen generoinnin (Myers, Sandler ja Badgett 2012, s. 38).

1.2 Kohdeorganisaation esittely

MIPRO Oy on vuonna 1980 perustettu raideliikenteen ja teollisuuden järjestelmiin erikoistunut yritys. MIPRO:n toimittamia järjestelmiä käytetään rautatie- ja metroliikenteen sekä teollisuusprosessien turvallisuuden hallinnassa. MIPRO toimittaa myös vesi- ja energiahuollon prosessien ohjausjärjestelmiä.

Kuva 2. Mipron toiminta-alueet

Yritys on rautatiejärjestelmien edelläkävijä ja markkinajohtaja Suomessa. Sen asiakkaita ovat esimerkiksi Väylä (Väylävirasto), HKL, Eesti Raudtee ja lukuisat vesi- ja energiahuollon yhtiöt.

MIPRO:n pääkonttori sijaitsee Mikkelissä ja sillä on toimipisteet myös Oulussa, Espoossa ja Tallinnassa. Työntekijöitä yrityksessä on tällä hetkellä noin 108. Liikevaihto vuonna 2020 oli noin 20 miljoonaa euroa.

Tässä tutkimustyössä keskitytään MIPRO:n rautatiepuolen ohjelmistotestauksen automatisointiin.

1.3 Testattavan tuotteen kuvaus

Mipro Oy kehittää ja toimittaa rautatieliikenteeseen turvalaitejärjestelmiä. Turvalaitteet ovat nykyään pitkälti automatisoituja järjestelmiä, joissa tietokoneet hoitavat ison osan niistä töistä jotka aiemmin jouduttiin tekemään käsin. Tämä on vähentänyt merkittävästi liikenteen ohjaamiseen tarvittavia resursseja sekä mahdollistanut tehokkaamman raideliikenteen käytön.

Turvalaitejärjestelmä koostuu asetinlaitteesta ja käyttöliittymäohjelmistosta. Asetinlaitteen tehtävä on varmentaa raiteilla tapahtuva turvallinen liikennöinti. Tietoliikenneyhteyksien välityksellä asetinlaite valvoo liikenteen sujuvuutta eri turvalaitteiden ja asetinlaittei-

den välillä. Se määrittelee turvalaitteiden loogisia tiloja ajon sallivasta ajon estäviin tiloihin sekä estää peräkkäisten kulkuteiden muodostumisen (Ratatekniset ohjeet osa 6 2014, luku 6.3)

Asetinlaitejärjestelmä muodostuu useasta asetinlaitteesta. Jokaisella niistä on rajatut toiminta-alueet, jotka on määritelty ratageometrian mukaan. Asetinlaitteet valvovat oman alueen turvalaitteiden tiloja ja kommunikoivat muiden asetinlitteiden kanssa. Asetinlaitetta voidaan käyttää erilliskäytössä (paikalliskäyttö) ja/tai kauko-ohjauksessa (Ratatieturvalaitteet 2014, s. 31-35).

Nykyiset asetinlaitekomponentit, -logiikat ja ohjaukset ovat sijoitettu radan varteen asetinlaitekaappeihin tai sitten erilliseen asetinlaitetilaan.

Seuraavissa kappaleissa esitellään lyhyesti keskeisimmät asetinlaitteen valvonnassa olevat turvalaitejärjestelmät joiden testaamisessa automatisointia tullaan hyödyntämään.

1.3.1 Opastimet

Opastimien tehtävän on ohjata raideliikennettä erilaisten symbolien välityksellä. Ne viestivät edessä olevien raideosuuksien tiloista ja junan sallitusta maksiminopeudesta. Opastimia on monenlaisia ja niitä kaikkia ohjaa kyseisen alueen asetinlaite. Seuraavassa lyhyesti yleisimmät käytössä olevat opastimet:

- Pääopastin

  Pääopastimen tehtävänä on näyttää junankuljettajalle alkavan kulkutien. Lisäksi sitä voidaan myös käyttää linjalla varmistetun suojastusosuuden aloittavana opastimena sekä myös junakulkutien päättävänä opastimena.

  Asetinlaite ohjaa pääopastimen toimintaa ja se määrittelee turvaehtojen pohjalta mitä opastin kulloinkin näyttää. Pääopastimella tiloja on kolme; Seis, Aja 35 sekä Aja. Aja 35km/h tarkoittaa raideliikenteessä maksiminopeutta jota juna saa ajaa seuraavalle opastimelle (Ratatieturvalaitteet 2014, s. 100-101).

Kuva 3. Pääopastimen tilat: Seis, Aja, Aja 35

- Esiopastin

Esiopastimen tehtävä on välittää ennakkotietoa seuraavan pääopastimen näyttämästä tilasta. Se on ainoa tieto minkä esiopastin voi välittää. Tiedolla pyritään ennakoimaan tulevia tilanteita sillä junilla on pitkät jarrutusmatkat eivätkä ne yleensä pysty pysähtymään näköetäisyydeltä. Esiopastimella on myös kolme erillistä tilaa:

- *Odota seis*, pääopastin näyttää Seis
- *Odota aja 35*, pääopastin näyttää Aja 35
- *Odota aja*, pääopastin näyttää Aja.

Esiopastin voi sijaita pääopastimen yhteydessä samassa mastossa tai erillisenä opastimena (Ratatieturvalaitteet 2014, s. 101).

Kuva 4. Esiopastimen tilat: Odota seis, Odota aja 35, Odota aja

➢ Raideopastin

Raideopastimella näytetään vaihtokulkutien opasteita. Raideopastimia käytetään pääsääntöisesti vaihtotyössä, mutta myös silloin kun ne ovat osa varmistettua junakulkutietä. Raideopastimella on myös kolme erillistä tilaa:

- *Seis*, opastinta ei saa ohittaa. Opastimen takana olevaa kulkutietä ei ole varmistettu eikä paikallislupaa ole annettu.
- *Aja varovasti*, kulkutie on varmistettu.
- *Ei opasteita*, opastin voidaan ohittaa liikenteenohjauksen myöntämän vaihtotyöluvan perusteella (Ratatieturvalaitteet 2014, s. 102).

Kuva 5. Raideopastimen tilat: Seis, Aja varovasti, Ei opasteita

➢ Suojastusopastin

Suojastusopastimen tehtävänä on opastaa, että raideosuudella on riittävät suojaetäisyydet. Sen avulla annetaan ennakkotietoa seuraavan suojastus- tai pääopastimen tiloista. Suojastusopastin ei kuitenkaan pysty välittämään pääopastimen Aja 35 opaste tietoa, jolloin tällaisia tapauksia varten linjalla on oltava erilliset esiopastimet. Kuten pääopastimilla, esiopastimilla ja raideopastimilla on myös suojaopastimella kolme erillistä tilaa:

- *Seis*, opastinta ei saa ohittaa. Suojavälin suojausehdot eivät täyty. Opastimen takana olevaa suojaväliä ei ole varmistettu.
- *Aja odota seis*, opastimen takana olevan suojavälin suojausehdot täyttyvät, mutta seuraava pää- tai suojastusopastin näyttää Seis – tilaa.

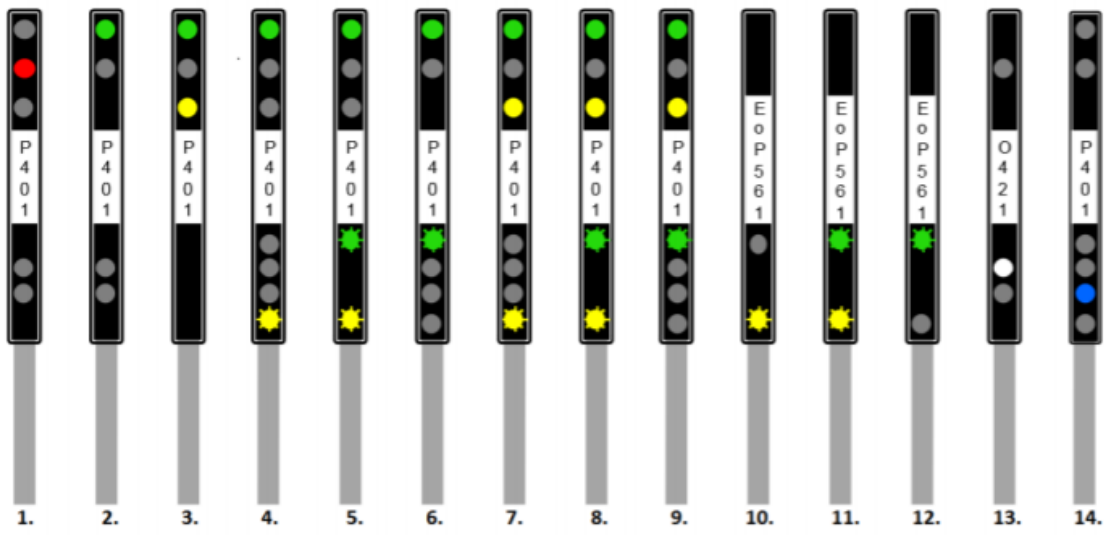
- *Aja*, opastimen takana olevan suojavälin suojastusehdot täyttyvät ja seuraava pää- tai suojastusopastin näyttää Aja – tilaa (Ratatieturvalaitteet 2014, s. 103).

Kuva 6. Suojastusopastimen tilat: Seis, Aja odota seis, Aja

➢ Yhdistelmäopastin

Yhdistelmäopastimella voidaan korvata yksittäisen pää-, esi- ja raideopastimen tai minkä tahansa edellä mainittujen opastimien kombinaation. Se voi olla junakulkutien tai vaihtokulkutien aloittava opastin, paikallislupa-alueeseen liittyvä opastin, ennakkotiedon antava opastin tai sivusuojan antava opastin (Ratatieturvalaitteet 2014, s. 104-105).

Yhdistelmäopastimelle on määritelty 14 erillistä tilaa, jotka on esitelty kuvassa 7.



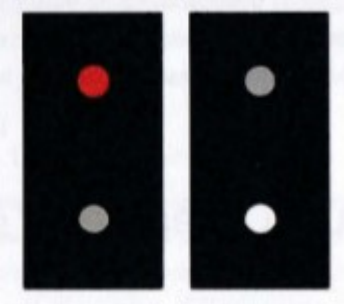
Kuva 7. Yhdistelmäopastimen tilat: 1. Seis, 2. Aja, 3. Aja 35, 4. Aja odota seis, 5. Aja odota aja 35, 6. Aja odota aja, 7. Aja 35 odota seis, 8. Aja 35 odota aja 35, 9. Aja 35 odota aja, 10. Odota seis, 11. Odota aja 35, 12. Odota aja, 13. Aja varovasti ja 14. Ei opasteita (Ratatieturvalaitteet 2014, s. 105).

- Lukitusopastin

Lukitusopastinta käytetään paikoissa, joissa raiteilla on laite tai järjestelmä, mikä saattaa estää liikennöinnin. Tyypillisesti tällaisia paikkoja ovat kääntöpöytä, portti tai kuormauspaikan kuormaus- tai purkulaite. Lukitusopastinta käytetään aina myös avattavan sillan yhteydessä. Lukitusopastimen opasteet ovat riippuvaisia vain siihen liittyvien laitteiden tilasta. Sillä ei esimerkiksi ole vaikutusta kulkutien aloittavan opastimen opasteeseen, mikäli kyseinen lukitusopaste sijaitsee kulkutiellä. Lukitusopastimella on kaksi erillistä tilaa:

- *Seis*, opastimeen liitetty laite on liikennöinnin estävässä asennossa. Opastin näyttää myös Seis – tietoa silloin kun laitteelta ei saada valvontatietoa.

- *Ei opasteita*, raiteella voidaan liikennöidä (Ratatieturvalaitteet 2014, s. 106).



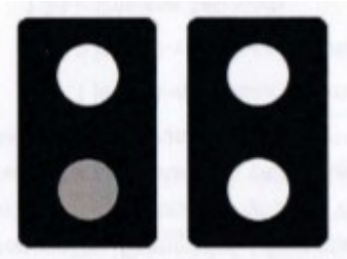
Kuva 8. Lukitusopastimen tilat: Seis, Ei opasteita

- Tasoristeysopastin

Tasoristeysopastinta käytetään rautateiden tasoristeyksien kuten teollisuuden- tai sivuraiteiden yhteydessä antamaan tietoa varoituslaitoksen hälytystilasta. Niitä ei käytetä pääraiteilla joilla kulkutie voidaan varmistaa. Varoituslaitoksen

tehtävänä on hälyttää tasoristeyksessä riittävän pitkään, että juna on tulossa. Tasoristeysopastimella ei voida antaa Seis opastetta eli yksikkö saa ajaa risteyksen yli silloin kun Lähesty varovaisesti – opaste on aktiivinen. Silloinkin nopeuttaa saa olla korkeintaan 10km/h. Tasoristeysopastimia ei saa asentaa raideosuuksille, joissa liikennöinti ylittää 35km/h. Tasoristeysopastimella on kaksi erillistä tilaa:

- *Lähesty varovaisesti*, varoituslaitoksen ehdot eivät täyty.
- *Ei opasteita*, varoituslaitos täyttää liikennöin sallivat ehdot; varoituslaitos hälyttänyt vaaditun ajan ja varoituslaitoksessa ei ole havaittuja vikoja (Ratatieturvalaitteet 2014, s. 107).



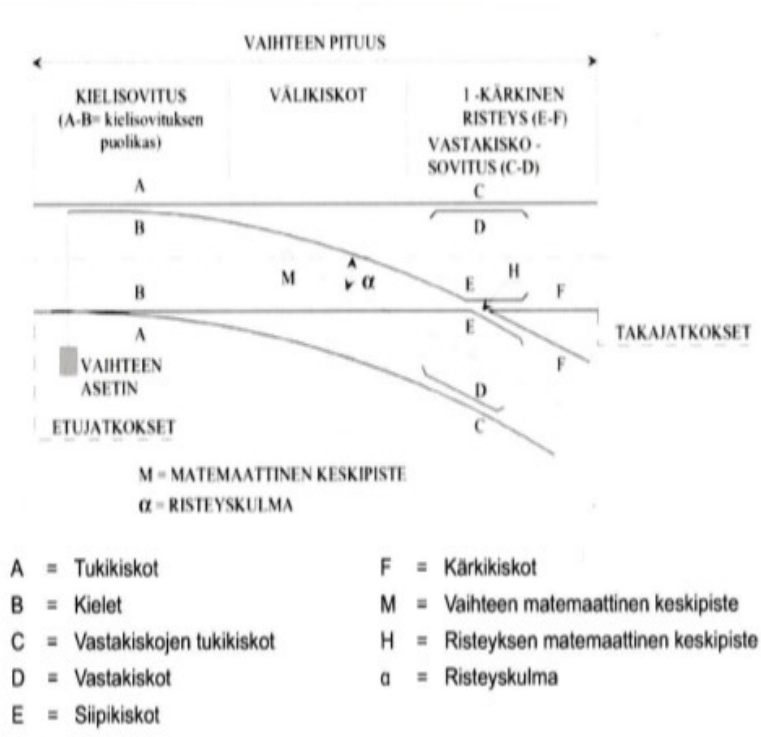
Kuva 9. Tasoristeysopastimen tilat: Lähesty varovaisesti, Ei opasteita

1.3.2 Vaihde ja vaihteen kääntölaitteet

Vaihde mahdollistaa raiteelta toiselle siirtymisen. Se on yksi tärkeimmistä komponenteista rautatiejärjestelmässä. Vaihde ohjaa yksikön raiteelta toiselle, rajoittaa nopeuksia vaihteen poikkeavalle raiteelle kuljettaessa ja on monimutkainen mekaaninen laite verrattuna esimerkiksi jatkuvaan kiskoon. Turvalaitteiden avulla ehkäistään vaihdeonnettomuuksien syntyminen.

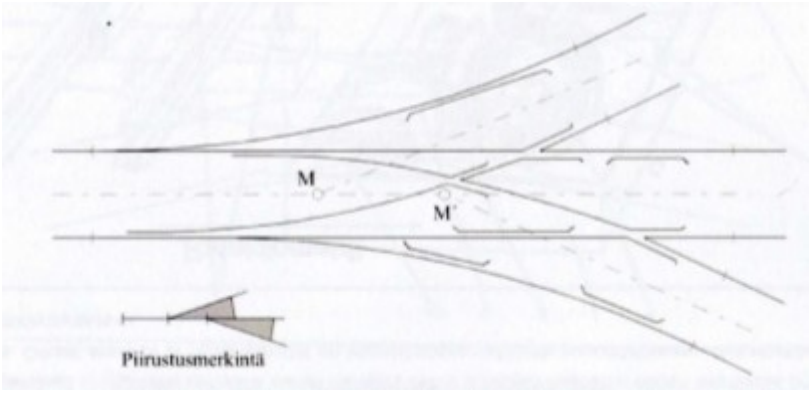
Suomessa on käytössä neljä erilaista vaihdetyyppiä. Vaihdetyypit määräytyvät vaihteen muodon ja poikkeavan raiteen sallitun nopeuden mukaan. Vaihdetyypit ja niiden yleisesti käytetyt lyhenteet (Ratatieturvalaitteet 2014, s. 108-110):

- Yksinkertaiset vaihteet (YV)



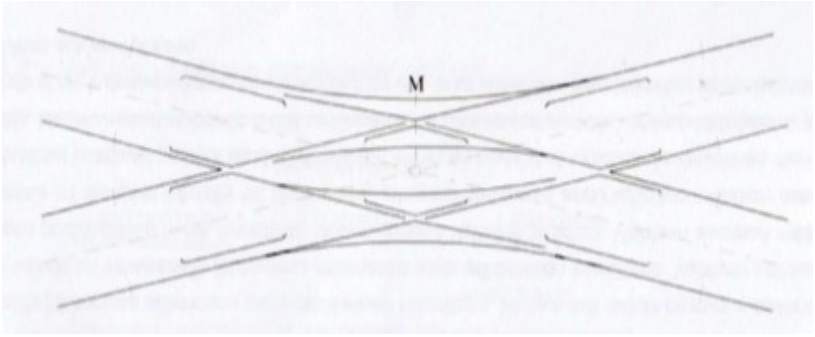
Kuva 10. Yksinkertaisen vaihteen rakenne ja pääosat

- Kaksoisvaihteet (KV)



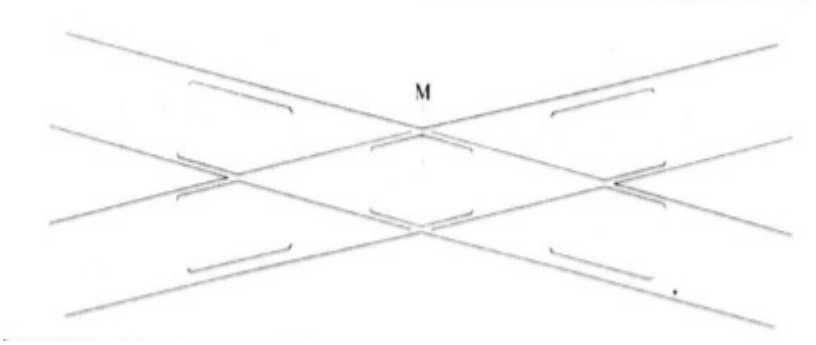
Kuva 11. Vasemmanpuolinen kaksisvaihde

- Yksipuoliset ja kaksipuoliset risteysvaihteet (YRV ja KRV)



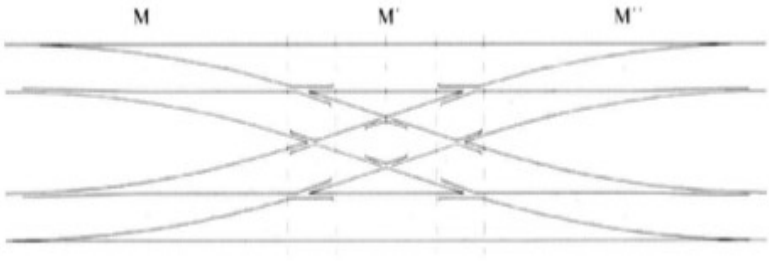
Kuva 12. Kaksipuolinen risteysvaihde

- Raideristeykset (RR)



Kuva 13. Raideristeys

- Sovitetut raideristeykset (SRR)



Kuva 14. Sovitettu raideristeys

Vaihteiden kääntölaitteet ovat sähköisiä ja mekaanisia. Sähkökäyttöisiä kääntölaitteita käytetään pääasiallisesti asetinlaitteelta käsin. Sitä voidaan käyttää myös asetinlaitteen käyttäjän antaman niin sanotun paikalliskäyttöluvan avulla paikallisesti ohjauspainikkeista. Mekaaninen vaihteen kääntö tehdään käsikammella, mihin tarvitaan paikalliskäyttölupa. Käsikammen käyttö on harvinaista ja sitä käytetäänkin vain erikoistilanteissa kuten vikatapauksissa; sähkökatkojen, kauko-ohjausvian yhteydessä tai kun kääntölaitetta ei ole vielä otettu käyttöön.

Sähköisillä kääntölaitteilla varustetut vaihteet lukittuvat aina automaattisesti, kun yksikön akseli oikosulkee vaihteen raidevirtapiirin. Tämä turvallisuuteen liittyvä toiminnollisuus estää vaihteen kääntämisen asetinlaitteelta käsin silloin kun juna on vaihteen päällä (Wikipedia: Rautatievaihde).

1.3.3 Raiteen sulku ja pysäytyslaite

Raiteen sulkulaitteen tehtävänä on suistaa yksikkö raiteelta tilanteessa jossa samalla raideosuudella on muuta liikennettä. Suistamisella vältetään todennäköinen yhteentörmäys. Suistaminen aiheuttaa aina kalusto- ja materiaalivahinkoja, mutta vahingot ovat huomattavasti pienemmät kuin yksiköiden yhteentörmäyksessä syntyvät vahingot. Raiteen sulkulaite on suunniteltu siten, että se voi pysäyttää hitaasti liikkuvan yksikön suistamatta sitä kiskoilta.

Kuva 15. Raiteen sulku ja sen käsikäyttöinen kääntölaite

Asetinlaitteen valvomilla alueilla raidesulun valvonta on liitetty asetinlaitteeseen. Raiteen sulkulaitetta käytetään antamaan sivusuoja kulkuteille tai paikallislupa-alueelle sellaisilla asetinlaitteen valvonnassa olevilla raideosuuksilla, joilla ei ole opastimia tai vaihdetta antamassa sivusuojaa (Ratatieturvalaitteet 2014, s. 120-121)

Pysäytyslaite toimii sivusuojana esimerkiksi ratapihoilla joissa se pysäyttää karkuun päässeitä yksiköitä. Pysäytyslaite on Suomessa ideoitu ja kehitetty turvalaite. Se on paljon edullisempi hankinta kuin turvavaihde. Pysäytyslaite toimii sitten, että karkuun päässeen yksikön ensimmäinen pyöräkerta nousee pysäytyskenkien päälle, jotka irtoavat kääntömekaniikasta. Jarruelementtien pohjaan on vulkanoitu kumimateriaali, jonka aikaansaama kitkavoima pysäyttää lopulta yksikön.

Pysäytyslaite voidaan kääntää kiskoille ja pois kiskoilta asettimella tai kääntölaitteella. Kääntölaitteella varustettua pysäytyslaitetta voidaan ohjata myös asetinlaitteelta erillisellä ohjauksella (Ratatieturvalaitteet 2014, s. 122-124).

1.3.4 Varmistuslukko ja avainsalpalaite

Varmistuslukolla ja avainsalpalaitteella lukitaan käsikäyttöinen vaihde, raiteensulku sekä valvotaan lukitun elementin asentoa. Varmistuslukko estää raiteensulun ja vaihteen kääntämisen silloin kun varmistuslukko on lukittu perusasentoon. Avainsalpalaitteen avulla voidaan valvoa raiteensulun ja vaihteen varmistuslukon käyttöavainta. Asetinlaite valvoo avainsalpalaitteiden tilatietoja (Ratatieturvalaitteet 2014, s. 124-128).

Kuva 16. Kaksi yksinkertaista varmistuslukkoa ja avainsalpalaitteen koteloa

2 Prosessimalli, standardit ja vaatimukset

Hyväksytyn toimintahäiriön raja **TFFR** (Tolerable Functional (unsafe) Failure Rate)	**Turvallisuuden eheyden taso** **SIL** (Safety Integrity Level)
$10^{-9} \leq TFFR \leq 10^{-8}$	4
$10^{-8} \leq TFFR \leq 10^{-7}$	3
$10^{-7} \leq TFFR \leq 10^{-6}$	2
$10^{-6} \leq TFFR \leq 10^{-5}$	1

Kuva 18. SIL asteikko vaarallisen vikaantumisen todennäköisyydelle (CENELEC, EN 50129). TFFR Tolerable Functional (unsafe) Failure Rate.

2.1 Standardit ja ohjeistukset

Mipron raideliikenteen turvalaitteet- ja ohjelmat perustuvat yleisesti raidelliikenteessä käytössä oleviin standardeihin. Standardisarja IEC 61508 on toiminnalliseen turvallisuuteen keskittyvä perusturvallisuusjulkaisu, johon perustuvat monet toimialakohtaiset toiminnallisen turvallisuuden standardit mukaan lukien raideliikennettä koskevat signaloinnin standardit EN 50126, EN 50128 ja EN 50129.

Kuva 19. Standardisarja IEC 61508.

Raideliikenteen toiminnallisen turvallisuuden signaloinnin eurooppalaiset standardit.

- *EN 50126 Osa 1 Yleinen RAMS (Reliability Availability Maintainability and Safety* - luotettavuus, saatavuus, ylläpidettävyys ja turvallisuus*) prosessi ja Osa 2 Järjestelmien lähestymistapa turvallisuuteen.*

  Tämä standardin Osa 1 tarjoaa rautatieliikenteen haltijoille ja rautatieyrityksille prosessin, joka mahdollistaa luotettavuuden, saatavuuden, ylläpidettävyyden ja turvallisuuden hallinnan, jota kutsutaan lyhenteellä RAMS. RAMS-vaatimusten määrittely- ja esittelyprosessit ovat tämän standardin kulmakiviä. Standardi edistää yhteisiä käsityksiä ja lähestymistapoja RAMS: n hallinnassa.

Kuten kuvasta 19 ilmeni EN 50126 on osa standardin IEC 61508 rautatieliikennesovellusta. Turvallisuuden osalta standardi EN 50126 Osa 1 tarjoaa turvallisuudenhallintaprosessin, jota tuetaan standardissa EN 50126 Osa 2 kuvattujen ohjeiden ja menetelmien avulla. EN 50126 Osa 1 ja 2 ovat riippumattomia käytetystä tekniikasta.

Turvallisuuden osalta EN 50126:ssa kuvaillaan turvallisuuden näkökulma toiminnallisella lähestymistavalla. Rautatieliikenteen haltijat ja rautatieyritykset soveltavat tätä eurooppalaista standardia järjestelmällisesti rautatieliikenteen elinkaaren kaikissa vaiheissa kehittäessään rautatiekohtaisia RAMS-vaatimuksia. Tämän standardin kehittämä järjestelmätason lähestymistapa helpottaa RAMS-vuorovaikutuksen arviointia rautatieliikenteen sovellusten elementtien välillä, vaikka ne olisivatkin luonteeltaan monimutkaisia.

Standardi edistää myös rautateiden sidosryhmien välistä yhteistyötä RAMS: n ja rautatieliikenteen kustannusten optimaalisen yhdistelmän saavuttamiseksi. Lisäksi standardi tukee Euroopan sisämarkkinoiden periaatteita ja helpottaa rautateiden yhteen toimivuutta Euroopassa (CENELEC, 2011, EN50126 Osa 1 s.7-9 ja EN50126 Osa 2 s. 6-9).

- *EN 50128 Rautatiesovellukset. Viestintä-, merkinanto- ja käsittelyjärjestelmät - Ohjelmistot rautatieohjaus- ja suojausjärjestelmille*

  EN 50128 määrittelee menettelyt ja tekniset vaatimukset ohjelmoitavien elektronisten järjestelmien kehittämiselle, joita käytetään rautatieohjaus- ja suojaussovelluksissa. Tämän standardin kansainvälinen versio on IEC 62279. Se on identtinen standardin EN 50128 kanssa.

  EN 50128 perustuvien eheiden ohjelmistojen kehittämisessä sovellettaviin periaatteisiin sisältyvät:

  - ylhäältä alas suuntautuvat suunnittelumenetelmät

  - modulaarisuus

  - kehityksen elinkaaren jokaisen vaiheen todentaminen

- todennetut komponentit ja komponenttikirjastot

- selkeä dokumentointi ja jäljitettävyys

- tarkastettava asiakirjat

- validointi

- arviointi

- kokoonpanon hallinta ja muutosten hallinta

- organisaation ja henkilöstön pätevyyskysymysten asianmukainen huomioon ottaminen.

Järjestelmän turvallisuusvaatimusten määrittely tunnistaa kaikki ohjelmistoille osoitetut turvallisuustoiminnot ja määrittää niiden järjestelmän eheyden tason (CENELEC, 2011, s.7-9).

- *EN 50129 Rautatiesovellukset. Viestintä-, merkinanto- ja käsittelyjärjestelmät - Turvallisuuteen liittyvät elektroniset merkinantojärjestelmät*

  EN 50129 standardia sovelletaan rautateiden merkinantosovellusten turvallisuuteen liittyviin elektronisiin järjestelmiin sisältäen osajärjestelmät sekä laitteet. Standardi koskee yleisiä järjestelmiä (geneeriset tuotteet tai järjestelmät, jotka määrittelevät sovellusluokan) sekä järjestelmiä erityissovelluksia varten.

  EN 50129 standardia sovelletaan kaikkiin turvallisuuteen liittyvän elektronisen järjestelmän elinkaaren vaiheisiin, keskittyen erityisesti arkkitehtuuriin ja järjestelmävaatimusten jakamiseen sekä järjestelmän hyväksyntään. Vaatimukset järjestelmille, jotka eivät liity turvallisuuteen, eivät kuulu tämän standardin soveltamisalaan (CENELEC, 2018, s.7 ja 21-22).

3 Kehitysvaiheen testausmenetelmät

Kehitysvaiheessa testaus seuraa V-mallin prosessia, missä aluksi yksittäiset ohjelman osat yksikkötestataan minkä jälkeen ne yhdistetään ja niille tehdään integrointitestaus. Integrointitestauksen jälkeen ohjelma on kokonaisuutena valmis järjestelmätestaukseen. Järjestelmätestauksen tarkastamisen ja hyväksymisen jälkeen ohjelma on valmis hyväksymistestaukseen. Kaikissa testauksen kehitysvaiheissa testaus voidaan tehdä monella eri tavalla. Tässä kappaleessa käydään läpi yleisimmin käytetyt testauksen tavat (Kasurinen 2013, s.64).

3.1 Black box – testaus

Black box eli musta -laatikko testaamisella tarkoitetaan testaamista, missä testaaja tietää miten ohjelman tulisi toimia. Testaaja tekee testitapaukset, missä kuvaillaan mitä testataan ja miten sekä mikä pitäisi olla testin lopputulos. Se mitä testaaja ei tässä testaustavassa tiedä on mitä ohjelmassa tapahtuu suorituksen aikana. Testaus onnistuu, jos määritelty lopputulos saadaan. Virheellisen lopputuloksen tapauksessa ohjelma palautuu takaisin kehittäjän korjattavaksi.

Black box –testaus on yleisimmin käytetty testausmuoto, koska stä voidaan käyttää kaikilla testauksen tasoilla; yksikkötestauksessa, integrointitestauksessa, järjestelmätestauksessa ja hyväksymistestauksessa (Kasurinen 2013, s.66).

3.2 White box – testaus

White box – testaus myös Glass box testauksena tunnettu testausmenetelmä eroaa black box testauksesta siten, että testaaja näkee koko ajan mitä ohjelma tekee missäkin vaiheessa suoritusta. Ohjelmalle annetaan erilaisia syötteitä, jotka käyvät ohjelman eri polkuja ja haaroja läpi sekä testaavat komentojen toiminnollisuuden (Khan, 2010).

White box testausta tehdään yleisimmin ohjelmiston koodausvaiheessa. Sen tekeminen edellyttää ohjelmointitaitoja sillä testauksessa tutkitaan ohjelman toimivuutta lähdekooditasolla. Testaajan on myös tunnettava testattava järjestelmä ja sen logiikka hyvin, että pystyy varmuudella todentamaan järjestelmän toimivuuden. White box testaus on syvällisempi ja tarkempi testausmenetelmä kuin black box testaus. White box testauksen heikkoutena voidaan todeta, ettei se havaitse esimerkiksi virheitä, jotka liittyvät huonosti tehtyihin vaatimusmäärittelyihin tai puuttuviin ominaisuuksiin. Joten kumpikaan black box

tai white box testausmenetelmä ei yksinään kata riittävän hyvin laadunvarmennustestausta (Kasurinen 2013, s. 67–68).

3.3 Gray box – testaus

Gray box testausmenetelmä eli harmaalaatikko testaus on nimensä mukaisesti sekoitus black box ja white box testaamisen menetelmiä. Siinä yhdistyy molempien menetelmien parhaimmat ominaisuudet kuten black box testauksen vaatimusmäärittelyihin perustuvien testitapausten testaus ja white box testauksen ohjelman toiminnollisuuden tutkiminen suoritustasolla.

Gray box testaaminen soveltuu hyvin esimerkiksi verkkopalveluiden testaamiseen, missä kaikkea testaamista ei pystytä kattavasti tekemään vain toisella testausmenetelmällä. Tällaisessa tilanteessa white box testausta käytetään verkkokaupan ydintoiminnallisuuden testaamisessa, missä tarkastellaan verkkokaupan ohjelman toimivuutta koditasolla ja kuinka sen eri komponentit toimivat. Black box testausta tarvitaan verkkokaupan liityntöjen ja rajapintojen testaamisessa. Testattavia asioita vois olla esimerkiksi palvelinyhteydet, maksupalvelut, erilaiset varmenteet ja muut ulkoiset sekä sisäiset liitynnät (Kasurinen 2013, s. 68).

3.4 Regressiotestaus

Regressiotestaus ei varsinaisesti ole erillinen testausmenetelmä vaan pikemminkin sillä tarkoitetaan uudelleentestaamista. Uudelleentestaamisella tarkoitetaan tässä yhteydessä tehtyjen ohjelmamuutosten jälkeistä testaamista, missä pyritään todentamaan ettei tehdyt muutokset ole rikkoneet mitään jo olemassa olevaa sekä aiemmin korjattuja ohjelman toiminnollisuuksia. Regressiotestauksella varmennetaan myös niin sanottujen väliversioiden toiminnollisuus, jotka sisältävät jo aiemmissa versiovaiheissa liitetyt uudet muutokset. Päämääränä on siis varmistua perustoiminnollisuuksien toimivuudesta.

Regressiotestauksen lähtökohtana onkin, että virheitä löytyy ohjelmaan liitettävistä uusista komponenteista ja toiminnollisuuksista, jotka ovat yhteydessä regressiotestattaviin komponentteihin. Siksi onkin tärkeää suhtautua kaikkiin muutoksin siten, että ne olisi kokonaan uudistettu. Muutoksia on tarkasteltava suurempana kokonaisuutena. Regressiotestausta hyödynnetään myös versionhallinnassa. Sillä todennetaan eri versiohaarojen yhdistämisen jälkeen, että kaikki aiemmissa versioissa korjatut osat toimivat myös yhdistetyssä versiossa.

Regressiotestaus on testauksen muoto, joka kannattaa automatisoida. Siinä käytettävien perustestitapausten tekeminen vie turhaa aikaa tutkivalta testaamiselta, testaaminen nopeutuu ja inhimillisten testausvirheiden määrä vähenee. Testausautomaatiota tarvitaan kehitystyön monessa vaiheessa esimerkiksi eri versioiden ja osatavoitteiden täyttymisen todentamiseen. Nämä testausvaiheet voivat toistua projektin aikana useaan otteeseen (Kasurinen 2013, s. 69-70).

3.5 Kuormitustestaus

Kuormitustestaus on testauksen vaihe mikä tehdään yleensä ennen tuotteen julkaisua. Siinä pyritään simuloimalla mittamaan ohjelmiston toimivuutta sen normaalilla käyttäjämäärällä ja käyttötavoilla. Oikeanlaisen testausympäristön – ja testien määrittely sekä konfiguraation käyttäminen ovat erityisen tärkeitä todenmukaisten kuormitustestitulosten saavuttamiselle.

Normaalin käyttäjäkuorman lisäksi kuormitustestausta käytetään ohjelmissa, verkossa tai laitteistossa olevien ongelmakohtien eli niin sanottujen pullonkaulojen paikantamiseen. Kuormitustestauksella pullonkaulat löytyvät helposti sillä ongelmatilanteiden toistaminen on helppoa. Lisäksi kuormitustestausta käytetään yleisesti maksimikuormituksen mittaamisen. Millä kapasiteetillä ohjelma toimii vielä normaalisti ennen kuin esiintyy merkittävää palvelun hidastumista, jumituksia, palvelun lukittumisia tai joitain muita epämääräisiä virhetilanteita. Syitä ongelmille voi olla monia kuten esimerkiksi palvelimien kuormanjako ei toimi suunnitellusti, palvelimien suorituskapasiteetti on alimitoitettu, verkkoko- ja/tai palvelin konfiguraatio ovat virheellisiä, ohjelmissa on aikasyöppöjä tai esiintyy muistivuotoja. Muistilevyjen kirjoitus- ja lukunopeudet voivat olla myös riittämättömiä.

Kuormitustestauksen äärimmäinen muotoa kutsutaan rasitustestaukseksi, missä testikuormaa lisätään yli ohjelmalle määritetyn maksimikapasiteetin. Testin päämäärä on, että ohjelman suoritus lopulta pysähtyy. Tämän kaltaisissa testeissä etsitään yleensä yhtäaikaisten kutsujen aiheuttamia lukituksia tai muistivuotoja. Rasitustestausta käytetään myös palvelunestohyökkäyksien testaamiseen.

Kuoritustestaus on lähes aina automatisoitua johtuen testisyötteiden isosta määrästä ja luonteesta. Siksi se kuormitustestaamiseen käytetäänkin erillisiä työkaluja. Manuaalista kuormitustestausta käytetään myös. Siinä käyttäjät päästetään kokeilemaan palvelua todellisuutta vastaavassa palvelinympäristössä (Beta -testaus), joka on skaalattu käyttäjäryhmän mukaan (Kasurinen 2013, s.71).

4 Testauksen tasot

4.1 Yksikkötestaus

Yksikkötestausta tehdään yleisesti aina ohjelmiston kehitysvaiheessa. Se on ensimmäinen ja myös tavallisin testauksen työmuoto ja menetelmä lähes kaikissa ohjelmistokehitystä tekevissä organisaatioissa. Yksikkötestauksessa testataan yhtä yksittäistä funktiota, moduulia tai olion toimintaa. Ohjelman tekijä testaa jatkuvalla syklillä tekemäänsä koodia. Yksikkötestauksessa tehdään myös ristiin testausta, missä toinen kehittäjä testaa toisen kehittäjän tekemää koodia. Näin saadaan laajempi kattavuus ohjelman toimivuuden testaamiseen.

Yksikkötestauksessa joudutaan usein rakentamaan joukko funktio- ja/tai oliokutsuja joihin ohjelman tulee pystyä vastaamaan ja suorittamaan kutsun toiminto. Yksittäiset ohjelman palaset eivät yksinään pysty suorittamaan mitään itsenäisesti. Niiden pitää olla vuorovaikutuksessa muiden ohjelman osien kanssa, jotta niiden toiminnollisuus pystytään testaamaan. Ohjelmiston sisäisten komponenttien pitää pystyä hakemaan tietoa tietokannasta ja kuuntelemaan verkon yli tapahtuvia kutsuja. Testien rakentamiseen meneekin paljon aikaa (Kasurinen 2013, s. 51-54), (Smart education, Jyväskylän Yliopisto).

4.2 Integrointitestaus

Yksikkötestausta seuraava testaustaso on integraatiotestaus, missä yksikkötestattuja ohjelman osia yhdistellään yhdeksi kokonaisuudeksi ja testataan niiden toimivuus lopulta yhtenä kokonaisuutena. Ohjelmien uudet osat integroidaan yleensä yksitellen jo aiemmin testattujen osien yhteyteen. Testausta varten voidaan joutua rakentamaan erilaisia tynkiä (*stub*), joiden tehtävänä on korvata sellaisia ohjelman osia joita ei vielä ole saatu valmiiksi, mutta joita tarvitaan, että integrointitestaus voidaan suorittaa.

Integraatiotestauksessa testataan suurempia kokonaisuuksia kuin esimerkiksi yksikkötestauksessa, mutta siinä ei kuitenkaan vielä testata täysin koko järjestelmää end-to-end – tyyppisesti sillä kaikki ohjelman osat eivät välttämättä ole vielä valmiita. Tyypillisiä integraatiotestausvaiheen testejä voisi olla eri ohjelman osien välisten yhteyksien ja viestien toimivuuden todentaminen ja samaa tietokantaa käyttävien ohjelman osien samanaikainen testaaminen.

Integraatiotestausta voidaan tehdä monella eri tavalla riippuen miten ja missä järjestyksessä ohjelmaan liitetään uusia ohjelman osia.

- Alhaalta ylöspäin integrointitavassa kaikista matalimman tason ohjelman osat integroidaan ensin. Osat jotka kommunikoivat suoraan raudan ja käyttöliittymän kanssa.

- Ylhäältä alaspäin integrointitavassa kaikista korkeimman tason ohjelman osat integroidaan ensin ja edetään hierarkiassa alaspäin.

- Voileipätestauksessa integraatiossa lähdetään ohjelmarakenteen molemmista päistä liittämään ohjelman osia yhteen yksi kerrallaan. Tässä vaiheessa voidaan säästää tynkien teossa, jos eri ohjelman osien liittämiset on hyvin jaksotettu. Testien yhteensovittaminen voi tämän testaustavan lopussa olla hankalaa.

Lopulta kun kaikki ohjelman osat on saatu onnistuneesti integroitua, voidaan ohjelman kehitysvaiheessa siirtyä järjestelmätestausvaiheeseen (Aili M. ja Fairouz Techier, Software Testing Consepts and Operations (E-book), s. 24), (Kasurinen 2013, s. 54-55).

4.3 Järjestelmätestaus

Yksikkötestauksen ja integraatiotestauksen jälkeen ohjelma siirtyy järjestelmätestaukseen. Ohjelma on tässä kehityksen vaiheessa koottu yhdeksi toimivaksi kokonaisuudeksi. Se ei sisällä enää mahdollisia integraation aikaisia tynkiä eikä muitakaan sijaiskomponentteja. Järjestelmätestaus voi sisältää kaikenlaista testausta kuten esimerkiksi tutkivaa testausta, kuormitustestausta tai käyttäjätestausta. Yleensä järjestelmätestaus mielletään kuitenkin työvaiheeksi, missä tehdään black box ja white box testausta.

Järjestelmätestaus tehdään siihen tarkoitetussa testiympäristössä. Testiympäristö ei ole lopullinen kohdeympäristö eli tuotantoympäristö vaan se on mahdollisimman samankaltainen kuin tuotantoympäristö. Järjestelmätestauksessa voi ilmetä vielä virheitä joita tarvitsee korjata kun taas järjestelmätestausta seuraavassa testaustaustasolla hyväksymistestauksessa vikoja ei pitäisi enää löytyä (Kasurinen 2013, s. 56-57).

4.4 HW testaus

HW (Hardware) eli rautatestauksessa testauksen pääpaino on komponenttien kytkentöjen testaaminen, tarkistaminen ja mittaaminen. HW testauksessa käytetään apuna erilaisia mittauskojeita ja laitteita. Testauksessa on apuna osaltaan myös ohjelmistotestauksen menetelmät, jossa esimerkiksi ohjelmallisesti tehdään joitain toimenpiteitä ja samanaikaisesti laitteistolle tehdään mittauksia tai muita testaustoimenpiteitä.

4.5 Hyväksymistestaus

Hyväksymistestaus on V – mallin testaustasoista viimeinen ennen käyttöönottotestausta. Hyväksymistestausvaiheessa virheitä ei enää etsitä eikä niitä saisi myöskään enää löytyä. Testauksessa painopiste onkin vahvasti toiminnollisuuksien todentamisessa. Hyväksymistestausympäristönä käytetään yleensä lopullista kohdeympäristöä.

Hyväksymistestauksen suorittaa yleensä tilaaja tai tilaajan valtuuttama yritys tai tarkastaja. Testauksen lopputuloksena ohjelmakokonaisuus joko hyväksytään tai hylätään. Hyväksymisen jälkeen ohjelmisto siirtyy juridisesti tilaajan omaisuudeksi ja sen kehityksenaikainen huolto- ja korjausvelvoite lakkaa olemasta voimassa (Kasurinen 2013, s. 57).

4.6 Käyttöönottotestaus

Käyttöönottotestauksen päätarkoituksena on varmistaa, että käyttöönotettu ohjelmakokonaisuus toimii lopullisessa käyttöympäristössä siten kuin se on suunniteltu.

Tämä testausvaihe tehdään heti hyväksymistestauksen jälkeen ja se sisältää yleensä samat testitapaukset kuin hyväksymistestauksessa. Ainoa ero hyväksymistestauksella ja käyttöönottotestauksella on, että käyttöönottotestauksessa testaus tapahtuu täysin toiminnassa olevassa tuotantoympäristössä ja hyväksymistestauksessa testausympäristö on rajatusti tuotantoympäristössä.

5 Testauksen dokumentaatio

Testauksessa syntyy paljon erilaista dokumentaatiota. Kattava testausdokumentaatio sisältää yleensä jokaisella testaustasolla testaussuunnitelman, testitapaukset, vikarapor-

tin ja testausraportin. Testausdokumentteja on mahdollista yhdistää suuremmiksi kokonaisuuksiksi riippuen projektien koosta. Pienemmissä projekteissa voi riittää yksi yleinen suunnitelma testauksesta sekä testausraportti, joka kattaa kaikki tasot. Suuremmissa projekteissa laaditaan yleensä niin sanottu Master Test Plan, mikä määrittelee raamit pienempien kokonaisuuksien testisuunnitelmille (Haikala ja Mikkonen 2011, 216-217).

5.1 Lähtötiedot

Testauksen suunnittelu ei pääse alkamaan ilman lähtötietoja. Lähtötiedot koostuvat normaalisti vaatimuksista, määrityksistä, käyttötapauksista, julkaisusuunnitelmista ja -tiedoista sekä muista ohjelman toiminnollisuutta määrittelevistä dokumenteista. Lähtötietojen pohjalta testaus saa tarvittavat tiedot siitä mitä järjestelmän tulisi tehdä kun sitä käytetään ennalta määritellyllä tavalla (Kasurinen 2013, s. 63).

5.2 Testaussuunnitelma

Projektitason testaussuunnitelma pohjautuu organisaation laatimaan testauspolitiikkaan ja testausstrategiaan.

5.2.1 Testauspolitiikka

Testauspolitiikan määrittelee organisaation ylin johto. Sen määrittämiseen ei tarvita syvempää tietämystä ohjelmistotuotantoon tai testaamiseen liittyvistä vaatimuksista.

Testauspolitiikka määrittelee yleensä:

- testauksen tavoitteet
- testausorganisaation
- testausprosessin, mitä testausta tehdään ja kuka määrittelee prosessin
- testaajien pätevyysvaatimukset
- noudatettavien standardien määritykset
- testauksen laadun mittarointimenetelmät

- testauksen tomintatapojen kehittämissuunnitelma

Testauspolitiikan onkin siis tarkoitus määritellä kuka tekee testausta ja miksi sitä tehdään.

5.2.2 Testausstrategia

Testausstrategia on suunnitelma, missä määritellään laajasti organisaation testaustoimintaa. Dokumentin laatii henkilöt joilla on tietämystä kuinka testausta tulisi tehdä ja kehitää. Heillä on käsitys mitä testaustyö kokonaisuudessaan on ja miten testausta on organisaatiossa aiemmin tehty.

Testausstrategia määrittelee muun muassa:

- yleisten riskien hallinnan
- testausken aloituskriteerit
- testusryhmän päätösvaltaisuuden eli mistä asioista testausryhmä saa tehdä itsenäisesti päätöksiä
- testausorganisaation tarkemmalla tasolla
- testausdokumentoinnin sisällön, laadinnan ja ylläpidon
- testausvaiheet
- käyttettävät testaustekniikat
- testitapausten valinnat ja priorisoinnit
- testiympäristön
- uudelleentestauksen kriteerit
- testauksen lopetuskriteerit

- poikkkeamien käsittelymenetelmät

5.3 Testitapaukset

Testitapaukset luodaan yleensä vaatimusmääritelmien, käyttötapausten ja arkkitehtuurimallien pohjalta. Testitapauksilla varmennetaan jonkin olemassaolevan, muutetun tai uuden toiminnallisuuden toimivuutta.

Testitapauksien yleiseen osioon kuvaillaan ympäristövaatimukset sekä versiot joilla testejä suoritetaan. Testityöksaluun jää myös aina jälki siitä kuka ja million testejä on kulloinkin luonnut ja kuka niitä on suorittanut.

Ennen testitapausten laatimista testaajan on tutustuttava tarjolla olevaan lähdemateriaaliin, jotta hän ymmärtäisi paremmin miten testattavan järjestelmän/ominaisuuden tulisi toimia. Muutoskorjausten tapauksessa on hyvä myös tietää miten jokin ominaisuus toimi ennen ja miten sen on tarkoitus toimia korjauksen jälkeen. Muutosdokumentaation tiedot auttavat testausta ymmärtämään paremmin muutettua toiminnallisuutta (Kasurinen 2013, s. 118-121).

Testitapauksen peruselementteinä voidaan pitää:

- Kuvataan testin lähtötilanne. Lähtötilanteessa voidaan määritellä ympäristö ja sen tila, lähtöparametrit sekä muut testin kannalta merkittävät riippuvuudet.
- Kuvataan mitä testissä tullaan tekemään askele (steppi) kerrallaan.
- Määritellään tavoiteltavat/toivotut testin tulokset jokaiselle testin stepille.
- Huomatuksiin kirjataan yleensä virhetapauksissa ongelman kuvaus.

Yleisesti virheitä löytyy tietyissä ohjelmistoon liityyvissä muutostilanteissa:

- Uuden ominaisuuden, koodin tai teknologian käyttöönotossa. Uusia asioita ei ole vielä aiemmin testattu, joten niistä ei ole aiempaa testaushistoriaa joihin suoritettavia testejä voisi verrata.
- Muutetun koodin testaamisessa.

- Viime hetken korjauksen tulos voi olla ns. hätäratkaisu esim. toimituasajan umpeutuessa. Ratkaisu ei aina ole kaunis eikä sitä kailta osin olla ennätetty testaamaan kattavasti.

- Ulkopuolelta tuodun koodin toimivuus ohjelmakokonaisuudessa on toimivuusriski, jos kaikkia rajapinta ja riippuvuussuhteita ei ole tarkkaan selvitetty. Näissä tilanteissa löytyy usein virheitä.

- Myös aiemmin toimivaksi todettu ja testattu ohjelmiston toimiminen uudella asiakkaalla voi olla ongelmallista. Tarkka järjestelmäkartoitus ja testaus on tässäkin tilanteessa tarpeellinen.

- Uudet työntekijät ja laitteet muodostavat mahdollisten virheiden syntymisen. Testaajan kokemus ja suhtautuminen testaamiseen vaikuttaa lopputulokseen kuten myös uusien laitteiden toimivuus ja luotettavuus.

Näiden lisäksi virheiden esiintymiseen voi vaikuttaa myös monet muut ulkoiset seikat kuten tulehtunut työilmapiiri, huono johtaminen, epämääräiset sekä muuttuvat sopimukset ja eri osapuolien kommunikointiongelmat (Kasurinen 2013, s. 118-121).

5.4 Vikaraportointi

Vikaraportointi on tärkeä vaihe ohjelmistokehityksessä. Sen oikeanmuotoisuus ja johdonmukaisuus vaikuttavat siihen kuinka hyvin korjaajat pystyvät korjaamaan havaittuja vikoja. Vikaraprtointi koostuu:

- Testiympäristön määrityksestä, versiotiedoista ja asetuksien määrityksistä

- Vian/ongelman kuvauksesta.

- Määritellystä ja toteutuneesta testituloksesta.

- Testivaiheiden kuvauksesta.

- Vian toistettavuudesta. Onko vika toistettavissa ja kuinka usein se esiintyy?

- Vian kriittisyyden määrittelystä (testaajan näkemys). Vian kriittisyysasteikko voi vaihdella matalan ja kriittisen välimaastossa.

Vikaraportin päätaarkoituksena on antaa mahdollisimman hyvät lähtötiedot korjaajalle vian korjauksen aloittamiseen. Kaikki testeissä havitut ilmiöt ja tehdyt toimenpiteet voivat auttaa korjauksen etenemisessä. Muun muassa testiajon nauhoituksesta on usein iso apu vian havannoinnissa (Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schröter, A. ja Weiss, C. 2010 s. 618-643).

Huonosti laadittu vikaraportti aiheuttaa väärinymmärryksiä, ylimääräisiä kyselyjä ja selvittelyjä, joskus jopa väittelyä ja muita ristiriitoja, jotka vievät aikaa ja resursseja. Oikeanlaiset lähtötiedot sekä sujuva kommunikointi toteutuksen ja testauksen välillä ovatkin yksi avainasioista toimivaan ohjelmistokehitykseen (Guo, Zimmermann, Nagappan & Murphy. 2011 s. 395-404).

5.5 Testausraportti

Eri testaustasojen kuten yksikkötestauksen, intgrointitestauksen, järjestelmätestauksen, HW-testauksen, hyväksymistestauksen ja käyttöönottotestauksen jälkeen laaditaan aina testausraportti.

Testausraporttiin kirjataan kaikki testaukseen liittyvät olennaisimmat asiat kuten:

- Mitä on testattu? Esitellään testauksen laajuus ja kattavuus.
- Kuka on testannut ja million?
- Missä ympäristössä ja millä ohjelmistoverioilla on testattu?
- Miten on testattu? Viittaus testaussuunnitelmaan.
- Avoinna olevien virheiden määrä, kuvaus sekä niiden luokittelu kriittisyyden mukaan.
- Testaamattomat testit
- Yhteenveto testeistä ja viittaus testien tarkempiin tuloksiin,

Testausraporttiin ei kuitenkaan ole tarkoitus sisältää matalan tason kuvauksia testitapauksista vaan testausraportin tehtävä on antaa kokonaistulos testatusta testaustasosta.

Testausraportteja voi myös luoda pienemmistä testauskokonaisuuksista, jotka ovat osa isompaa testattavaa kokonaisuutta. Tällöin projektissa muodostuu useampia testausraportteja joista koostetaan projektin lopussa kokonaistestausraportti (Koomen T., Pol M. 1999 s. 143).

6 Automatisoinnin toteutus

6.1 Yleistä

Automatisoinnin päätarkoituksena oli vähentää kehityksen ja etenkin testauksen manuaalisen työn määrää sekä nopeuttaa testausprosessia ja osaltaan myös parantaa testauksen laatua vähentämällä inhimillisten virheiden määrää. Automatisointi mahdollistaa testaajien kohdistaa voimavaransa tutkivaan testaamiseen, jonka tekeminen ei niinkään ole ollut ajallisesti mahdollista aiemmin.

Turvalaitejärjestelmien testaamiseen liitty paljon eri osapuolien tekemiä testejä, verifiointeja ja validointeja monessa eri kehityksen vaiheessa. Testimassojen jatkuvasti kasvaessa automatisointi katsottiin tuovan helpotusta moniin näihin havaittuihin asioihin.

6.2 Robot Framework

6.2.1 Yleistä

Robot Frameworkkin kehitys sai alkunsa Pekka Klärckin diplomityöstä vuonna 2005 (Eliga Oy, Pekka Klärck diplomityö) ja ensimmäinen kehitysversio Robot Frameworkista julkaistiin samana vuonna Nokia Networksilla. Tällä hetkellä viimeisin julkaistu versio on 3.1.1 tammikuulta 2019 (Wikipedia, Robot Framework).

Robot Framework on avoimeen lähdekoodiin pohjautuva testiautomaatio –kehys, joka on julkaistu Apache License 2.0: lla. Kehystä pääsääntöisesti käytetään hyväksymistestaukseen, hyväksymistestausvetoiseen ohjelmistokehitykseen (ATDD, Acceptance test-driven development) sekä robottiprosessiautomaatiossa (RPA).

Robot Frameworkissa käytetään testien kuvaamisessa avainsanapohjaista rakennetta (KDT, Keyword-driven testing). Menetelmä tunnetaan myös nimellä toimintasanapohjainen testaus. Siinä erotetaan testitapausten dokumentointi testien suorittamistavasta.

Robot Framework on helposti laajennettavissa. Se voidaan integroida melkeinpä mihin tahansa muuhun työkaluun tehokkaiden ja muunneltavien automaatioratkaisujen luomiseksi. Robot Framework ei myöskään maksa käyttäjälle mitään, koska se pohjautuu avoimeen lähdekoodiin missä lisenssimaksuja ei ole. Se on käyttöjärjestelmästä ja sovelluksesta riippumaton. Siinä ydinkehys on toteutettu Pythonilla ja toimii myös Jythonilla (JVM) ja IronPythonilla (.NET).

Robot Frameworkissa on käyttäjäystävällinen syntaksitekniikka, jossa käytetään selokikielisiä avainsanoja. Sen ominaisuuksia voidaan laajentaa kirjastoilla, jotka voidaan toteuttaa Pythonilla tai Java: lla. Robot Frameworkin kehys sisältää vakiokirjastoja ja joukon työkaluja testien muokkaamiseen, ajamiseen ja niiden kehittämiseen. Kirjastot sisältävät avainsanoja, jotka tarjoavat todelliset automaatio- ja testausominaisuudet testaukseen. On myös olemassa erikseen kehitettyjä ulkoisia kirjastoja, jotka voidaan tarpeen mukaan asentaa. Uusien kirjastojen luominen onnistuu myös kätevästi.

Robot Frameworkin GitHubissta löytyy kehyksen lisätietoja, lähdekoodia ja ongelmien seurantaa liittyviä asioita (Robot Framework).

6.2.2 Esimerkki

Seuraavassa esimerkki, jossa testataan selainsovelluksen sisäänkirjautumisen kelvollisuutta.

- Avainsanat

  Testitietojen syntaksit perustuvat avainsanoihin, jotka ovat koostettavissa kuvan 20 esimerkin kaltaisesti. Uudet avainsanat voidaan määrittää siten, että ne käyttävät

aiemmin luotuja avainsanoja. Tällä tavalla voidaan abstraktoida testauksen yksityiskohdista jotain järkevää; esimerkiksi käyttäjän ei tarvitse tietää mitä tarkalleen vaihe Submit Credentials todellisuudessa tekee. Siksi testitapaukset ovat selkeitä, luettavissa ja ennen kaikkea ymmärrettäviä. Oikean abstraktiotason avulla saadaan välitettyä testin todellinen tarkoitus.

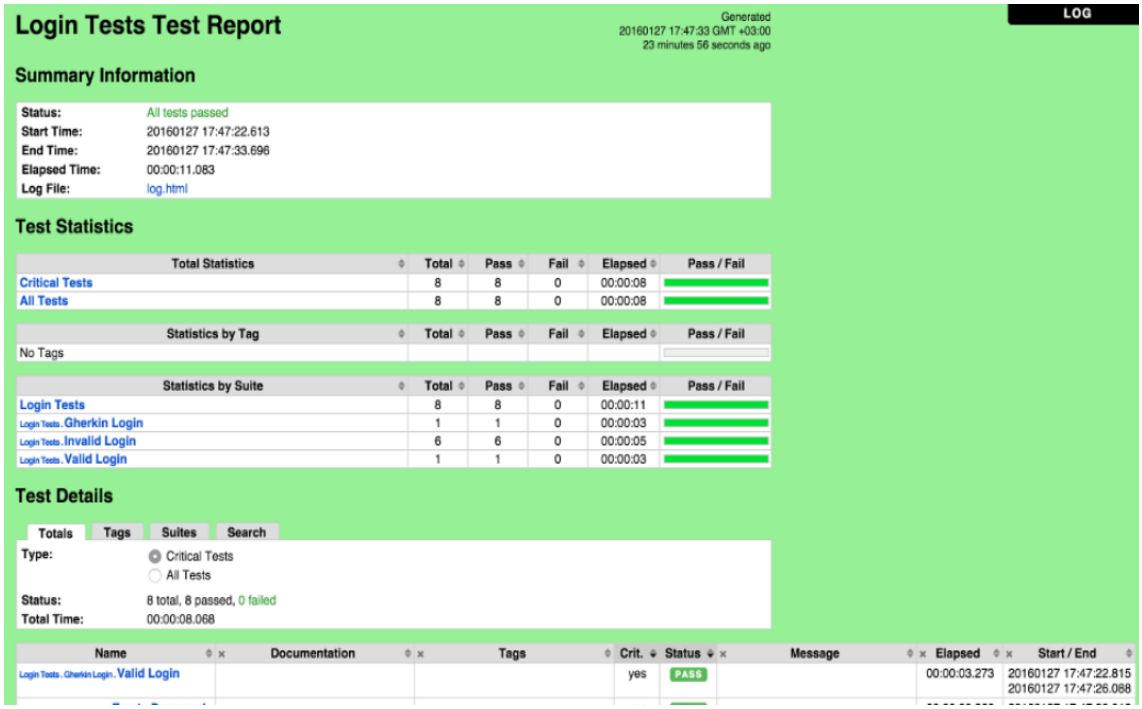
```
*** Settings ***
Documentation     A test suite with a single test for valid login.
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          resource.txt

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Kuva 20. Testin avainsanat

- Testiraportit

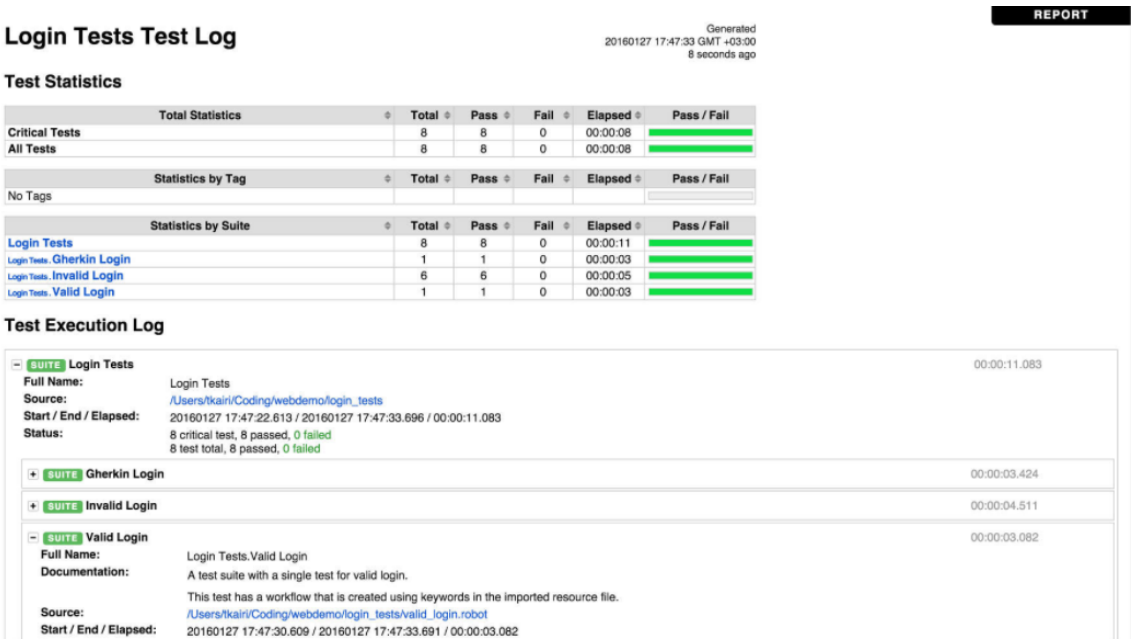
Robot Framework muodostaa ajetuista testeistä raportin ja lokitiedoston. Raportista näkee onnistuneiden ja hylätyiden testien tiedot sekä niiden suoritusajat. Kuvan 21 testiraportti on Kuvan 20 testitapauksesta.



Kuva 21. Robot Framework testiraportti

- Testiloki

  Testin lokitiedoista näkee yksityiskohtaisesti testien jokaisen vaiheen, avainsanoista avainsanoihin. Lokissa näkyy myös hylättyjen testien virhekohdat. Kuvassa 22 esimerkki testin lokitiedot.



Kuva 22. Robot Framework testiloki

- Testikirjastot

  Kaikki testeissä käytettävät avainsanat ovat peräisin testikirjastosta. Testikirjastossa on vakioavainsanoja ja sinne on mahdollista luoda uusia avainsanoja tarpeen mukaan. Kuvassa 23 esimerkki laskintestin kirjastosta (Robot Framework).

```
from calculator import Calculator, CalculationError

class CalculatorLibrary(object):
    def __init__(self):
        self._calc = Calculator()
        self._result = ''

    def push_button(self, button):
        self._result = self._calc.push(button)

    def push_buttons(self, buttons):
        for button in buttons.replace(' ',''):
            self.push_button(button)

    def result_should_be(self, expected):
        if self._result != expected:
            raise AssertionError('%s != %s' % (self._result, expected))

    def should_fail(self, expression):
        try:
            self.push_buttons(expression)
        except CalculationError, err:
            return str(err)
        else:
            raise AssertionError("'%s' should have failed" % expression)
```

Kuva 23. Robot Framework testikirjastot

Robot Framework käyttää Selenium kirjastoa, joka soveltuu selainkäyttöisten käyttöliittymien testaamiseen. Kuten Robot Framework pohjautuu myös Selenium avoimeen lähdekoodiin ja siksi se soveltuukin erityisen hyvin selainpohjaiseen testiautomaatioon (Simon Stewart, 2009).

7 Automatisoinnin jalkauttaminen organisaatioon

7.1 Käyttöönotossa huomioitavat yleiset asiat

Käyttöönoton edellytyksenä on, että yrityksen toimintamallit ja tavat mahdollistavat automatisoinni toteuttamisen siten, että yritys saa siitä tavoiteltua hyötyä.

Käyttöönotosta tulee olla hyvä toteutussuunnitelma, jotta sillä olisi edellytykset onnistua. Käyttöönottoa edeltää yleensä pilotointivaihe, missä koko automatisointiprosessia koeponnistetaan pienimuotoisessa ja yleensä lyhytkestoisessa muutaman viikon tai kuukauden kestävässä projektissa. Pilotointivaiheesta kerätyllä tiedolla rakennetaan

käyttöönoton suunnitelmaa. Suunnitelmaan korjataan pilotointivaiheessa havaitut putteet ja kehityskohdat. Näin saadaan rakennettua hyvä infrastruktuuri automatisoinnin toteutukselle (Frewster ja Graham, 1999 s.293-294).

7.1.1 Työvälineet ja testausympäristö

Työvälineiden käyttöönotossa on myös tärkeää huomioida yrityksen strategiat ja säännöt. Työvälineiden kuin myös koko automatisointi prosessin dokumentointi, määritykset ja ohjeistukset tulee olla selkeitä ja kaikkien käytettävissä.

Testausympäristön asentamisen, ylläpidon ja käytön ohjeistukset tulee määritellä siten, että niillä saavutetaan mahdollisimman vakaa ja hallittava testausympäristö. Testausympäristö on eristetty muista ohjelmistoympäristöistä. (Frewster ja Graham, 1999 s.293-294).

7.1.2 Koulutukset

Automatisointityön tekemisen opettelemiseen on varattava riittävästi aikaa ja resursseja. Usein tämän vaiheen tärkeyttää arvioidaan virheellisesti. Ei ajatella, että opeteluun kuluisi paljon aikaa ja resursseja. Käyttäjille tulee antaa riittävästi aikaa uuden menetelmän opetteluun, muuten automatisoinnista ei saada tavoiteltua maksimihyötyö käyttöön (Frewster ja Graham, 1999 s.293).

7.1.3 Resurssointi

Työvälineiden, ohjelmien ja työmenetelmien sovittaminen olemassaolevien töiden rinnalle tuo omat haasteensa (Kettunen ja Simons, 2001 s.21). Uusien automatisoitavien testien luomiseen sekä myös testien ylläpitoon ja kehittämiseen on varattava henkilöitä ja aikaa. Automatisointi vaatii yleensä aina alkuvaiheessa enemmän resursseja, jotta se saadaan kunnolla käyntiin. Sitten kun testeillä on saatu katettua tavoiteltu osuus testauksesta, voidaan automatisointiin varattua resurssointia vähentää.

7.2 Käyttöönotto

Automatisoinnin käyttöönotossa päätavoitteena on, että automaattinen testaus olisi helpompaa, nopeampaa ja laadukkaampaa kuin käsin tehtävä testaus.

Käyttöönotto tapahtuu käyttöönottosuunnitelman mukaisesti ja sen toteutumista seurataan eri mittareilla ja arvioinneilla (Frewster ja Graham, 1999 s.293-294).

7.3 Käyttöönoton välittömät hyödyt ja haitat

Automaatisoidut testit alkavat tuottamaan toistokoertojen mukaan hyötyä yritykselle noin kahdeksannen toistokerran jälkeen. Hyödyt kasvaa exponentiaalisesti testikertojen kasvaessa. Lisättyjen uusien testien määrä kasvattaa hyötysuhdetta.

Automatisointiin investoidut varat eivät vielä käyttöönoton alussa maksa itseään takaisin vaan takaisinmaksu tapahtuu sen mukaan miten hyvin automatisointi saadaa yrityksessä jalkautettua. Siihen vaikuttaa muun muassa:

- miten hyvin automatisointi on saatu sopimaan yrityksen toimintamalleihin ja menetelmiin
- testiympäristön ja työkalujen toimivuus
- työntekijöiden pätevyys

8 Tulosten mittarointi

8.1 Yleistä mittaroinnista

Mittaroinnilla pyritään saamaan parempi ymmärrys mittauksen kohteena olevasta asiasta tai ilmiöstä. Mittausten avulla päästään tekemään tarkempia johtopäätöksiä ja toimenpiteitä. Mittaus on aina perusluonteeltaan objektiivista eli on riippumaton mittaajan arvoista ja sen yleiset säännöt noudattavat kaikilla tieteenaloilla samaa perusperiaatetta. Mittaristo koostuu yhdestä tai useammasta mittarista. Mittaristo koostuu yleensä siten, että olemassa olevaan mittaristoon on yhdistetty uusia mittareita tai se voi perustua johonkin yksilöityyn mittaristomalliin tai -viitekehykseen. (Lönnqvist, Kujasivu, Antikainen, 2006 s.13-32).

8.2 Mittaroinnin vaatimukset

Mittarit on johdon työkalu joita se käyttää päätöksenteossaan. Mittareiden antamien arvojen perusteella johto tekee tarvittavia ohjausliikkeitä toimintojen parantamiseksi. Niiden tarkoitus on siis helpottaa päätöksentekoa.

Päätöksenteko voidaan vaiheistaa esimerkiksi:

- mittaraiden määrittämiseen
- mittaustulosten tuottamiseen
- mittaustulosten analysointiin ja määrittämiseen
- tulosten hyödyntämiseen päätöksentekoprosessissa
- lopullisen päätöksen tekemiseen

Päätöksen teko koostuu siis monesta vaiheesta, jotka yhdistyvät mitattuun tietoon ja sitä kautta yrityksen tuloksiin. Tietojen on siis täytettävä ennalta sovitut vaatimukset, että luotettava päätöksenteko olisi mahdollista (Laitinen 2003, s.145–147).

8.3 Mittariston suunnittelu

Mittareiden suunnittelu on syytä tehdä huolella sillä väärin asetetuista mittareista ei ole mitään hyötyä pikemminkin niistä voi koitua yritykselle yllättäviä kustannuksia. Mittareiden huolellinen suunnittelu ja valinta yhdessä mittareiden pilotoinnin kanssa luovat hyvän perustat hyödyllisten mittareiden käyttöönottolle. Mittaroinnin vaatimukset ja määritelmät tulevat yleensä johdolta. Johto määrittelee yrityksen arvomaailman ja strategian mukaissesti mitä mitataan ja miksi. Sen tulee huomioida myös, ettei mittarointia rakenneta liian suppeaksi mikä saattaa johtaa vääriin tulosten tulkintoihin (Kankkunen., Lehtinen. ja Matikainen. 2005, s.116-119).

8.4 Mittaroinnin hyödyt ja haitat

Mittarit antavat selkänojaa päätöksentessa ja niiden pohjalta korjausliikkeiden teko on luotettavaa, tehokasta ja helppa. Ilman mittareita tuloksilla ei ole vertailukohdetta ja päätöksien teko on pohjautuu arvioihin, ennustuksiin ja oletuksiin.

Mittarit toimivat eri työntekijäryhmissä eri tavalla. Työntekijöille on tärkeää, että työ on mukavaa, tuloksellista ja palkitsevaa. Yritysten bonusohjelmat ovat sidottuina työssä saavutettuihin päämääriin. Tämä osaltaan kannustaa työntekijöitä tavoitteellisuuteen yhdessä työssä viihtymisen kanssa. Esimiesten sekä muu operatiivinen johdon on saatava tietoa nopeasti muutuuvista tilanteista, jotta he ennättävät tekemään suunnitelmiin tarvittavia muutoksia. Yrityksen keskijohdon tehtävä on seurata tulosten kehitystä ja arvioida niiden pitkäaikaisvaikutusta. Ylimmän johdon tehtävä on seurata tulosten kokonaiskuvaa ja sitä miten ne mukautuvat yrityksen strategiaan.

Mittarointi ohjaa yritystä keskittymään toiminnan kannalta sen tärkeimpiin asioihin. Sen avulla vältytään myös turhien ja päällekkäisten asoiden tekemiseltä (Etelälahti 2019, 83–84).

Oikeanlaisen mittaroinnin löytäminen ja suunnittelu voi viedä paljon aikaa. Myös mittaroinnin muuttaminen voi olla työlästä. Mittareiden sovittamien työntekijöiden normaalin työn oheen tuo omat haasteensa. Totuttujen työrutiinien rikkominen vaatii yleensä aina totuttelua, jotta uuteen työskentelymalliin taas tottuu. Jatkuvasta mittareiden muuttamisesta voi kehittyä jo rasite mikä saattaa näkyä jo mittariston tuottamissa tuloksissakin.

Vajavaisella tai virheellisellä mittaustiedolla voi olla välittömiä negatiivisia seuraamuksia. Nopeasti muuttuva liiketoiminta tuo omat haasteensa mittareiden kehittämiselle. Mittareiden jatkuva tarkkailu ja niiden muuttaminen tavoitteitaan palvelevampaan muotoon on ratkaisevaa yrityksen menestyksen kannalta (Kankkunen., Lehtinen. ja Matikainen. 2005, s.19-30).

9 Yhteenveto ja pohdintaa

Opinnäytetyön tavoitteena oli selvittää miten ohjelmistotestauksen automatisointi toteutettiin kohdeyrityksessä. Automatisointi saatiin onnistuneesti toteutettua työn kuvailemalla tavalla. Sen käyttöönotto- ja mittarointivaiheet olivat opinnäytetyön valmistuessa vielä käynnissä ja siksi kyseisiä osa-alueita ei tässä työssä ennätetty enempää käsittelemään. Automatisoinnin merkeittäviä hyötyjä voitiin kuitenkin nähdä jo heti sen käyttöönoton ensimmäisessä projektissa. Testejä jotka ensimmäisenä automatisoitiin sisälsivät paljon yksinkertaisia toistettavia toimintoja. Niitä oli lukumäärällisesti paljon ja ne saatiin suhteellisen pienellä työmäärällä automatisoitua. Niiden automatisointi vapautti testaajien aikaa tutkivan testaamisen tekemiseen.

Yhtenä työn tavoitteena oli myös kertoa lukijalle mitä ohjelmistotestaus on, mistä se koostuu ja selvittää miten tarpeellista on testauksen automatisointi. Työssä esitellyllä testaustyön kuvauksella aiheesta tietämätön lukija pystyy paremmin ymmärtämään myös automatisoinnin tärkeyden sekä sen tarpeellisuuden.

Opinnäytetyön aihe oli mielenkiintoinen ja sitä tehdessä opin paljon uusia asioita automatisoinnista sen suunnittelusta ja toteutuksesta. Erityisen mielenkiintoista oli se kuinka kuinka automatisointi saatiin käynnistettyä Mipro Oy:ssä, mistä lähdettiin liikkelle ja mihin päädyttiin. Kuinka koko prosessi saatiin pala palalta suunnitelmallisesti koottua yhteen ja käyttööotettua. Käyttöönoton ja mittaroinnin tulokset ja kuvaukset eivät valitettavasti tähän työhön ennättäneet sillä ne olivat opinnäytetyön kirjoittamisen päättyessä vielä keskeneräiset.

Teoriatasolla työn laatiminen ei ehkä olisi ollut niin jännittävä ja mieleenpainuva. Jatkossa tulen seuraamaan kuinka käytönoton laajentaminen onnistuu Mipron muissa projekteissa ja millaisila mittareilla automatisointia tullaan seuraamaan sekä mittareiden tuloksia.

Lähteet

Aili M. ja Fairouz Techier, Software Testing Consepts and Operations (E-book). https://ebookcentral.proquest.com/lib/metropolia-ebooks/detail.action?docID=4040909&query=test+automation

Banerjee K. 2018. Test Automation – A Recipe for Continuous Delivery Success. Luettu 6.5.2018. https://dzone.com/articles/test-automation-a-recipe-for-continuous-delivery-s

CENELEC EN 50129, SIL asteikko vaarallisen vikaantumisen todennäköisyydelle. https://www.en-standard.eu/ilnas-en-50129-railway-applications-communication-signalling-and-processing-systems-safety-related-electronic-systems-for-signalling-1/

Feitelson D. G., Frachtenberg, E. ja Beck, K. L. 2013. Development and Deployment at Facebook. IEEE Internet Computing. Volyymi n:o 17. Lehti numero 4, s. 8-17

Haikala, I. & Mikkonen, T. 2011, Ohjelmistotuotannon käytännöt, Talentum, Helsinki.

Kasurinen J. 2013. Ohjelmistotestauksen käsikirja

Kasurinen J. 2015 Luento: Ohjelmistotestauksen perusteet: https://docplayer.fi/5479942-Ct60a4150-ohjelmistotestauksen-perusteet-jussi-kasurinen-etu-suku-lut-fi-kevat-2015.html

Khan M.E. 2010. Different Forms of Software Testing Techniques for Finding Errors. IJCSI, Vol 7, Issue 3, No 1, May 2010.

Myers G., Sandler C., & Badgett T. 2012. The Art of Software Testing. 3. painos. Hoboken, New Jersey: John Wiley & Sons, Inc. S. 38.

Wikipedia, V-maali. https://fi.wikipedia.org/wiki/V-malli

Wikipedia, Rautatievaihde. https://fi.wikipedia.org/wiki/Rautatievaihde

Smart education, Jyväskylän yliopisto. http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot

Ratatekniset ohjeet osa 6. 2014. Väylän ohjeita. http://www2.liikennevirasto.fi/julkaisut/pdf8/lo_2014-07_rato6_web.pdf

Rautatieturvalaitteet 2014, Viitanen J ja Järvinen L. Liikennevirasto 2014.

What Makes a Good Bug Report? - Software Engineering, IEEE Transactions , Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schröter, A. & Weiss, C. 2010.

Not My Bug!" and Other Reasons for Software Bug Report Reassignments. Proceedings of the ACM 2011 conference on Computer supported cooperative work 2011, Guo, P. J., Zimmermann, T., Nagappan, N. & Murphy, B.. ".

A practical step-by-step guide to structured testing, Koomen T. and Pol M., Test Process Improvement: Addison-Wesley 1999.

EN50126 Railway Applications - Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 1: Generic RAMS Process, CENELEC, 2017

EN50126 Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 2: Systems Approach to Safety, CENELEC, 2017

EN50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, CENELEC, 2011

EN50129 Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling, CENELEC, 2018

Eliga oy, Pekka Klärck diplomityö. http://eliga.fi/writings.html

Wikipedia, Robot Framework. https://fi.wikipedia.org/wiki/Robot_Framework

Robot Framework. https://robotframework.org/

Simon Stewart, Google Open Source Blog. 2009. https://opensource.googleblog.com/2009/05/introducing-webdriver.html

Frewster M. ja Graham D. 1999, Software Test Automation: Effective use of test execution tools. London: Addision-Wesley

Kettunen J. ja Simons M. 2001. Toiminnanohjausjärjestelmän käyttöönotto pk-yrityksessä. Vantaa, VTT.

Lönnqvist Antti, Kujasivu Paula ja Antikainen Riikka. 2006. Suorituskyvyn mittaaminen - tunnusluvut asiantuntijaorganisaation johtamisvälineenä. Helsinki: Oy Nord Print Ab.

Kankkunen Kari, Lehtinen Lasse ja Matikainen Esa. 2005. Mittareilla menestykseen. Jyväskylä: Gummerus Kirjapaino Oy.

Etelälahti Pekka. 2019. Tulosmittauksen pieni käsikirja. BoD Books on Demand, Helsinki.

Laitinen, Erkki K. 2002. Yritystoiminnan uudet mittarit, 3. Jyväskylä: Gummerus Kirjapaino Oy.