Ville Hietala

Usean käyttäjän ICT-testausympäristön järjestelyt

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mobile Solutions

Insinöörityö

10.5.2022

Tiivistelmä

Tekijä:	Ville Hietala
Otsikko:	Usean käyttäjän ICT-testausympäristön järjestelyt
Sivumäärä:	28 sivua
Aika:	10.5.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Sektorijohtaja Jouni Ruotanen Lehtori Toni Spännäri

---

Insinöörityössä tutkittiin nykyaikaisen usean käyttäjän ICT-testausympäristön toimintaa ja laitteiston vaatimuksia. Työssä tehtiin myös kartoitusta nykyaikaisien kehitysmenetelmien ja etätyön vaikutuksesta testausympäristön toimintaan ja vaatimuksiin.

Tietoa kerättiin teemahaastatteluilla, haastattelemalla Puolustusvoimien yhteisen integraatio- ja tuotekehitysympäristön PVITY:n käyttäjäprojektien avainhenkilöitä, kuten projektipäälliköitä ja järjestelmäpäälliköitä. Haastatteluista poimittiin toistuvia huomioita, ja niiden perusteella tehtiin päätelmiä mahdollisista tarpeista ja parannuskohteista. Haastatteluista selvisi, että nykyaikaisia menetelmiä kannattaa hyödyntää ja niillä, etenkin etätyöllä, nähtiin mahdollisia etuja esimerkiksi resurssien tehokkaampaan käyttöön. Etäyhteyksien käytön nähtiin tehostavan henkilöresurssien käyttöä, kun testausta voidaan suorittaa ympäri maata eikä aina tarvitse kuljettaa testaajia PVITY:n testaamaan.

Huomioitaviksi parannuskohteiksi erottuivat alemman turvaluokan testit mahdollistava testidata ja ohjeistuksen uudistuksista tiedottamisen parannukset. Lisäksi huomattiin, että etätyönä tehtävä testaaminen saattaa kuitenkin vaatia paikan päällä olevia tukitoimia ongelmatilanteiden ratkomiseen. Myös tietoturva huomioitiin tärkeänä ja osittain myös rajoittavana tekijänä testauksessa.

Insinöörityön tuloksia voidaan käyttää päivitettäessä PVITY:n testaajan ohjetta ja suunniteltaessa tiedottamista, varauskäytäntöjä ja mahdollisia tulevia kehitysprojekteja.

Avainsanat: testausympäristö, laboratorio, tietoturva

Abstract

Author:	Ville Hietala
Title:	Organising a multi-user ICT-testing environment
Number of Pages:	28 pages
Date:	10 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information technology
Professional Major:	Mobile Solutions
Supervisors:	Jouni Ruotanen, Sector Manager
	Toni Spännäri, Senior Lecturer

The purpose of the thesis study was to research the requirements for a multi-user ICT-testing environment. The effect of remote work and modern development practices was also mapped. The information was gathered by interviewing the key users of Finnish Defense Forces product development and integration testing environment PVITY.

The method used in the interviews was a semi-structured interview. The interviews were recorded, and notes were taken. The answers of the interviews were compared, and the main points were added to a table where similarities between interviews were color coded. From the results of the deduction, it was possible to see the main points that the interviewees agreed upon. Based on the results, some suggestions were made for future improvements in PVITY.

Based on the interviews it was concluded that modern practices of product development are mostly welcomed, but it was seen that there could be some issues with the processes of FDF in taking them to use. Also working remotely was seen as an opportunity to enhance productivity and the usage of testing capacity.

The results of this research can be used in updating the PVITY user guide and when planning the future development of PVITY.

Keywords: Testing environment, laboratory, security

Sisällys

Lyhenteet

ICT: *Information and communications technology*. Tieto- ja viestintätekniikka.

SAFe: Scaled Agile Framework for enterprises. Suurille yrityksille räätälöity ketterä ohjelmistokehitysmenetelmä.

PVITY: Puolustusvoimien yhteinen integraatiotestaus- ja tuotekehitysympäristö.

CI: Continuous Integration. Jatkuva integraatio. Tuotekehitysmenetelmä, jossa ketteryyden ja testiautomaation avulla on saatu nopeutettua muutosten tuomista tuotantoympäristöön.

Katakri: Kansallinen turvallisuusauditointikriteeristö. Ulkoministeriössä toimivan kansallisen turvallisuusviranomaisen laatima tietoturvaohjeistus.

1 Johdanto

Insinöörityön tarkoituksena on tutkia nykyaikaisen ICT-testausympäristön järjestelyjä ja kehitystä nykyaikaisien kehitysmenetelmien ja tietoturvan kannalta. Insinöörityössä haastatellaan Puolustusvoimien yhteisen kehitys- ja integraatioympäristön, PVITY:n, käyttäjiä ja haastattelujen pohjalta laaditaan toimenpide-esityksiä, joita voidaan käyttää Puolustusvoimien kehitysympäristöjen suunnittelun lähtökohtina.

Insinöörityön alussa perehdyttiin erilaisiin tutkimusmenetelmiin ja tuotekehitysprosesseihin. Työssä päädyttiin käyttämään tutkimusmenetelmänä laadullista tutkimusmenetelmää, teemahaastattelua, johon valittiin pääkysymys, miten toteuttaa usean käyttäjän ICT-testausympäristön järjestelyt, ja 7:ää tarkentavaa lisäkysymystä työn tilaajan kanssa sovituista aiheista, kuten tietoturvan ja ketterien menetelmien vaikutuksesta testausympäristön suunnitteluun.

Haastatteluihin pyydettiin PVITY:n eri käyttäjäprojektien avainhenkilöitä, haastattelut tallennettiin ja analysoitiin merkitsevimmät vastaukset koodaamalla kysymyskohtaisesti yhtenevät vastaukset eri haastatteluista. Yhtenevien vastausten perusteella laadittiin toimenpide-ehdotuksia, jotka esitellään Puolustusvoimien henkilöstölle.

2 Tutkimus

2.1 Tutkimusmenetelmät

Perinteisesti tutkimusmenetelmät on jaettu kolmeen eri tutkimusstrategiaan, kokeelliseen-, kysely- ja tapaustutkimukseen [1, s. 122]. Kokeellinen tutkimus soveltuu erityisen hyvin tilanteisiin, joissa pystytään mittaamaan jonkin muuttujan vaikutusta toisiin muuttujiin ja koe on helposti toistettavissa. Määrällisessä kyselytutkimuksessa laaditaan kyselylomake, jonka perusteella kartoitetaan tietyn ihmisjoukon suhdetta tutkittavaan asiaan. Jokaiselta tutkimukseen osallistuvalta

kysytään samat asiat ja niihin tulee ennalta määritellyt vastausvaihtoehdot. Tapaustutkimuksessa tutkitaan yksityiskohtaisesti jotain yksittäistä asiaa. Tapaustutkimusta on ennen sen nykyistä määrittelyä kutsuttu tieteilijöiden piirissä ”kenttätyöksi” [1, s. 123], mutta sittemmin on vakiintunut nimitys kvalitatiivinen eli laadullinen tutkimus.

Tässä työssä tutkimusmenetelmäksi valittiin laadullinen tutkimus, johon kerätään materiaalia alan kirjallisuudesta sekä asiantuntijoiden teemahaastatteluilla. Tutkimusmenetelmän valintaan vaikutti eniten potentiaalisten haastateltavien rajallinen määrä, minkä vuoksi kyselytutkimuksen tuottama tieto ei olisi ollut kovinkaan merkittävää.

2.2 Laadullinen tutkimusmenetelmä

Laadullinen, eli kvalitatiivien, tutkimus on tutkimusmenetelmä, jossa tutkimuksen tekijä hankkii tietoja tutkittavasta aiheesta, esimerkiksi tutkimalla alan kirjallisuutta, omilla havainnoilla tai tekemällä haastatteluita. Laadullista tutkimusta pidetään yleisesti määrällisen, eli kvantitatiivisen, tutkimuksen vastapuolena. Laadullista tutkimusta pidetään pehmeänä tutkimustapana, koska se tuottaa subjektiivisempaa tietoa kuin määrällinen tai luonnontieteiden tutkimus [2, s. 21]. Laadullisen tutkimuksen tuloksissa näkyy ilmiöiden prosessimaisuus, joten laadullisen tutkimuksen tulokset ovat usein ajasta ja paikasta sekä tutkijasta riippuvaisia. Tutkimuksen tekijän oma lähtökohta, mikäli sillä on merkitystä, on syytä mainita tutkimuksen lähtökohdissa [1, s. 278–279].

2.3 Tutkimuksen rajaukset, reunaehdot ja kysymykset

Tässä työssä tutkitaan testausympäristöjä yleisellä tasolla. Työssä pyritään selvittämään, miten määräytyvät testausympäristön tekniset minimivaatimukset. Pohditaan, miten uudet tekniikat muuttavat testausta ja testausympäristöjä. Lisäksi selvitetään tieto- ja kyberturvallisuuden roolia testauksen ja testausympäristön suunnittelussa.

Tutkimus rajataan koskemaan testausympäristöjen kehitystä ja ympäristöjen kehityksen näkymiä.

Tutkimuskysymyksiksi valittiin seuraavat kysymykset:

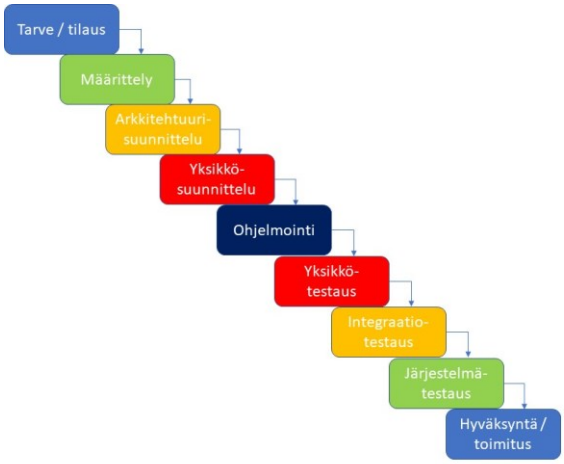
- Miten järjestää testausympäristön konfiguraationhallinta, jotta erilaiset testaustarpeet saadaan parhaiten sovitettua?
- Minkälaisia vaatimuksia on laitteistolle hyvässä testausympäristössä?
- Mitä testausympäristön tietoturvassa pitää ottaa huomioon? Esimerkkeinä suojaukset, käyttöoikeudet ja tietoturva.
- Minkälainen rooli virtualisoinnilla ja testiautomaatiolla on nykyaikaisessa ICT-testausympäristössä?
- Miten jatkuva integraatio (CI) ja ketterät menetelmät vaikuttavat testiympäristön suunnitteluun?
- Miten voi toteuttaa usean käyttäjän testausympäristön hallinnan? Esimerkkinä testien aikatauluttaminen.
- Miten etätyö vaikuttaa testausympäristön käyttöön, suunnitteluun ja ylläpitoon?

3 Ohjelmistojärjestelmän kehitys

3.1 Järjestelmäkehityksen vaiheet

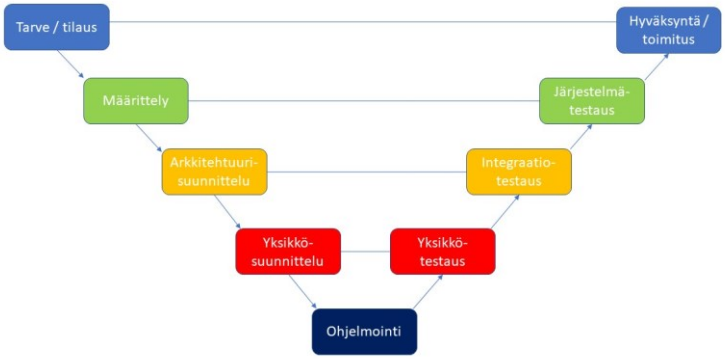
Järjestelmien kehittäminen tehdään vaiheittain [3, luku 2.1]. Ensin päätetään, mitä tarvitaan, ja määritellään, mitä ominaisuuksia järjestelmällä pitää olla. Seuraavana vaiheena on suunnitella toteutustapa. Tätä vaihetta kutsutaan arkkitehtuurisuunnitteluksi [3, luku 2.5.4; 4]. Arkkitehtuurisuunnitteluvaiheessa suunnitellaan esimerkiksi järjestelmän eri osien väliset rajapinnat [4]. Arkkitehtuurisuunnittelusta edetään suunnittelemaan yksittäisiä järjestelmän osia, mitä voidaan kutsua yksikkösuunnitteluksi tai moduulisuunnitteluksi. Kun yksiköt on saatu suunniteltua, voidaan aloittaa toteutus, käytännössä ohjelmointi. Ohjelmointi voi pitää sisällään varsinaisen uuden ohjelman kirjoittamisen ja myös olemassa olevan ohjelmiston asetusten määrittelyn.

Järjestelmäkehitysprosessia on usein kuvattu vesiputousmallikaavion avulla, jossa kehityksen vaiheet seuraava toisiaan vesiputouksen kaltaisesti (kuva 1).



Kuva 1. Testauksen vaihejako ja vesiputousmalli [4].

Uudempi tapa kuvata prosessia on V-malli (kuva 2), jossa toisiaan vastaavat vaiheet on laitettu v-muodon vastapuolille. V-mallilla pystytään paremmin havainnollistamaan, mitä suunnittelun vaihetta missäkin testausvaiheessa testataan.

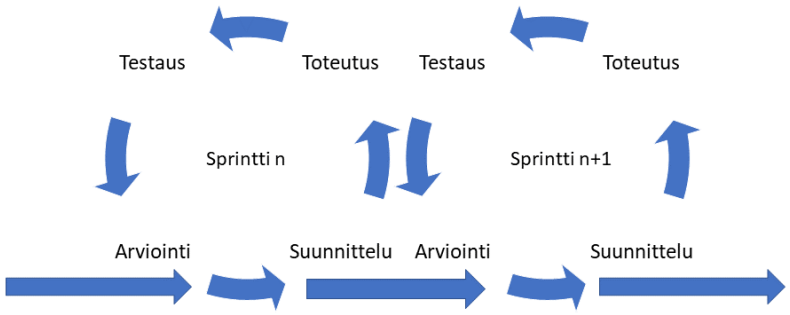


Kuva 2. Testauksen vaihejako ja V-malli [3, luku 2.5.3; 4].

3.2 Ohjelmistokehitys käyttäen ketteriä menetelmiä

Vaihtoehdoksi perinteiselle vesiputousmallille on kehitetty uudenlaisia menetelmiä, joilla pyritään tekemään ohjelmistokehityksestä joustavampaa ja nopeampaa reagoimaan muutoksiin. Esimerkkinä näistä uudenaikaisista kehitysmenetelmistä ovat niin kutsutut ketterät kehitysmenetelmät. Ketterillä menetelmillä pyritään vähentämään kehitysprojektien kankeutta ja byrokratiaa. Ketterässä ohjelmistokehityksessä työ jaetaan pieniin nopeasti toteutettaviin osiin, jolloin projektin johto pystyy paremmin seuraamaan kehityksen edistymistä sekä kohdistamaan resursseja paremmin sinne, missä niitä tarvitaan. Ketterillä menetelmillä voidaan myös rajata epäonnistumisen riskejä.

Ketterissä menetelmissä kehitystä tehdään lyhyissä kehitysjaksoissa, joiden aikana tuotettua tulosta kutsutaan inkrementiksi. Iteraatio koostuu useammasta muutaman viikon mittaisesta sprintistä, jonka aikana suoritetaan ennalta sovitut tehtävät [5, s. 11–13]. Jokainen sprintti sisältää ennalta määritellyt vaiheet: ensin saadaan tavoitteet ja suunnitellaan, sitten toteutetaan, seuraavaksi testataan ja lopulta tarkastellaan, miten hyvin tulos vastaa tavoitteita (kuva 3).



Kuva 3. Sprinttien rakenne-esimerkki.

Ketterissä menetelmissä ovat siis mukana ne vaiheet, jotka ovat myös perinteisissä menetelmissä. Samat kehityksen vaiheet toistuvat jokaisessa sprintissä

pienemmälle osalle ohjelmistoa kuin perinteisissä kehitysmenetelmissä. Näin toimittaessa ongelmat voidaan huomata varhaisemmassa vaiheessa. Tunnetuimpia ketteristä menetelmistä ovat scrum ja kanban. [5.]

Kanban

Kanban on ketterä kehitysmenetelmä, jossa painotetaan tehtävien edistymisen esittämistä havainnollisesti [5, s. 3]. Kanban-prosessissa tehtävät asetellaan taululle, jossa on sarakkeet ainakin seuraaville asioille: lista tulevista tehtävistä, parhaillaan työstettävistä tehtävistä ja valmiista tehtävistä (kuva 4). Jokainen projektin tehtävä, myös testaus, suunnitellaan kanban-taululle omana korttinaan, ja kun tehtävä poimitaan suoritettavaksi, sen kortti siirretään tehtävän tilaa vastaavaan sarakkeeseen. Kanban-kortit sijoitetaan sarakkeisiin niin, että kriittisimmät kortit ovat ylimmäisinä ja vähemmän merkitykselliset alempana. Kanban-korttien perusteella voi projektin johto saada kuvan projektin etenemisestä. Kanban-menetelmässä iteraation kestoa ei ole rajattu. [5.]

Tehtävälista	Työn alla	Valmis
Koodi 4	Koodi 2	Koodi 1
Testi 3	Koodi 3	
Testi 4	Testi 1	
	Testi 2	

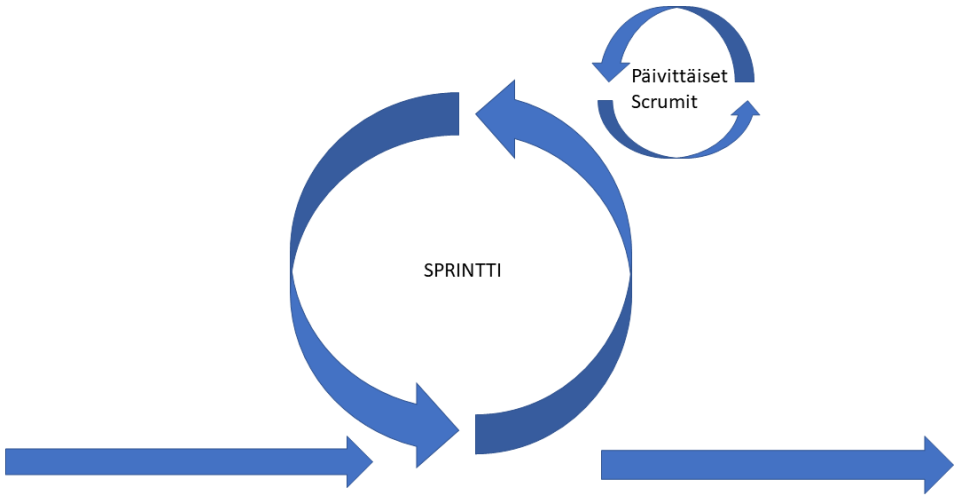
Kuva 4. Esimerkki kanban-taulusta.

Scrum

Scrum on ketterä kehitysmenetelmä, jossa kehitysprojekti jaetaan helposti hallittaviin osiin.

Scrumissa projektin osallistujat jaetaan yhteen tai useampaan scrumtiimiin, joissa jokaisessa on kolme erilaista roolia. Scrum-roolit ovat tuotteen omistaja, kehitystiimi ja scrummaster. Yksi ryhmän jäsen toimii tuotteen omistajana, joka vastaa tiimin työn arvon maksimoimisesta. Scrummaster valvoo scrum-menetelmän käyttöä ja huolehtii, että tiimin jäsenet noudattavat scrum-kehityksen menetelmiä ja käytäntöjä. Scrummaster myös tarkkailee työn edistymistä ja tarvittaessa hoitaa kommunikaatiota tuotteen omistajan ja kehitystiimin välillä siitä, miten mahdolliset ongelmatilanteet korjataan. Kehitystiimissä voi olla useita jäseniä, jotka suorittavat itse varsinaista tuotekehitystä. [5.]

Scrumissa työtä tehdään sprinteissä (kuva 5), jotka ovat tyypillisesti 2–4 viikon mittaisia [6, s. 32].



Kuva 5. Scrum-sprintin rakenne [6, s. 33].

Scrumtiimi kokoontuu päivittäin scrummasterin johdolla käymään läpi edellisen 24 tunnin saavutukset, tulevan 24 tunnin tehtävät ja mahdolliset työn edistymistä haittaavat haasteet. Päivittäisten tilannekatsausten avulla scrummaster saa kuvan projektin edistymisestä ja voi kohdentaa resursseja sinne, missä

apua tarvitaan [6, s. 31]. Scrumissa koko ryhmä on vastuussa tuotteen laadusta ja näin myös siitä, että testaus tulee suoritetuksi.

DevOps ja jatkuva kehitys

Kun tuotekehitys on nopeutunut ja palvelut siirtyneet internetin pilvipalveluiksi, on tullut tarve kehittää myös tuotekehityksen prosesseja. 1970- ja -80-luvuilla tuotekehitys kesti usein vuosia ja jos jotain meni vikaan, riskit koskivat koko yrityksen tulevaisuutta. 1990-luvulla palvelut muuttuivat asiakas-palvelintyyppisiksi ja kehitysaika lyheni vuosista kuukausiin, enää ei koko yritys ollut vaarassa, mutta hinta oli edelleen korkea. Tähän ratkaisuksi on kehitetty DevOps-menetelmä, jossa tuotteeseen voidaan tehdä nopeasti pieniä muutoksia, ilman että epäonnistumisen hinta on merkittävä [7, s. xxii–xxiii].

DevOps vaatii, että koko tuotekehityksen ajatusmaailma muutetaan tukemaan sitä. DevOps:iin liittyy olennaisesti ketteryys, jatkuva kehitys ja testiautomaatio. DevOps:n tavoitteena on lyhentää aikaa, joka menee siitä, kun kehittäjä saa koodin valmiiksi, siihen, että päivitys on asiakkaalla. DevOps:n avulla kehittäjä voi tuottaa useita päivityksiä päivässä, ja aina kun saa jotain valmiiksi, sovellukselle ajetaan ennalta määritellyt, kattavat testit. Tällä automatisoidulla testauksella pystytään saamaan kiinni viat nopeasti ja epäonnistunut päivitys ei päädy tuotteeseen ilman korjausta. DevOps:n avulla voidaan siis rajata riskiä. [7, s. 11–13.]

Scaled Agile Framework

Scaled Agile Framework (SAFe) on kokoelma ohjeita, joiden perusteella ketteriä menetelmiä voi skaalata toimintaan suurten yritysten tasolla. SAFe antaa työkalut siihen, miten suunnitellaan ja johdetaan ketterää työskentelyä. SAFe pohjautuu vanhempiin ketteriin menetelmiin, kuten leaniin ja Agile Manifestiin. SAFe antaa työkaluja yritysten toiminnan saamiseksi ketteräksi, mutta se myös vaatii yrityksen johdolta ketterää suhtautumista toimintaan. [8.] SAFe:ssa keskiössä on Agile Release Train, jota voidaan kutsua esimerkiksi julkaisujunaksi [9]. Julkaisujunassa on useampia kehitystiimejä ja niiden tukitiimit. Julkaisujunalla

rytmitetään tiimien kehitystä niin, että toisistaan riippuvien tiimien välinen toiminta helpottuu [9]. SAFe:a kehittää Coloradossa toimiva yritys Scaled Agile Inc. yhdessä laajan useiden toimijoiden yhteisön kanssa [10].

3.3 Testaus

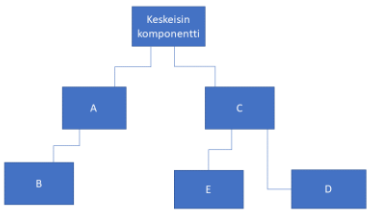
Valmistuneelle ohjelmistolle pitää suorittaa testaus, jotta sen toiminnasta voidaan varmistua. Ensimmäisenä on vuorossa yksittäisen ohjelmistokomponentin testaaminen eli yksikkötestaus. Yksikkötestauksessa tutkitaan, toimiiko tehty ohjelmistokomponentti määrittelynsä mukaisesti. Kun on saatu toteutettua useita komponentteja, ne pitää yhdistää toimivaksi järjestelmäksi. Ohjelmiston kokoamisprosessia kutsutaan integroinniksi. Integroinnin yhteydessä tehdään integraatiotestaus, jossa testataan yksiköiden välisiä rajapintoja. Integraatiotestauksessa paljastuvat rajapintojen suunnittelun virheet. [4.] Lopulta suoritetaan järjestelmätestaus, jossa testataan, toimiiko järjestelmä määrittelynsä mukaisesti.

Yksikkötestaus

Yksikkötestauksessa testataan yksittäistä ohjelmiston komponenttia ja sen toimintaa. Siinä pyritään paljastamaan ohjelmoinnissa ja yksikkösuunnittelussa tapahtuneet virheet. Yksikkötestausta voidaan kutsua myös moduulitestaukseksi tai komponenttitestaukseksi. Yksikkötestit suunnitellaan yleensä testattavan ohjelmistokomponentin määrittelyn pohjalta. Yksikkötestaus on riippumaton muista ohjelmiston osista, siksi sen voi suorittaa, vaikka muut komponentit eivät olisi vielä valmiita. Jos komponentti käyttää yhteyksiä muihin ohjelmistokomponentteihin, näitä yhteyksiä korvaamaan tehdään simuloitu ympäristö, josta käytetään nimeä testipeti. Moduulitestaus on poikkeuksellinen testauksen taso, koska se on ainoa testausvaihe, jossa on järkevää tutkia ohjelman lähdekoodia. Tällaista testimenetelmää kutsutaan white box -menetelmäksi. [3, luku 3.1; 4.]

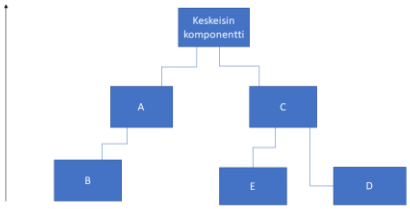
Integraatiotestaus

Integraatiotestauksessa testataan ohjelmistokomponenttien välistä toimintaa (kuva 6). Siinä paljastuvat arkkitehtuurisuunnittelun virheet. Integraatiotestauksessa testataan lähtökohtaisesti järjestelmään lisätyn yksittäisen komponentin ja siihen liitettävien komponenttien välisiä rajapintoja [3, luku 5.2]. Integraatiotestaus toteutetaan yleensä black box -menetelmällä, eli testauksessa ei katsota komponenttien sisältämää lähdekoodia. Ohjelmistokomponenttien integroimisessa järjestelmään voidaan käyttää erilaisia menetelmiä. Keskeisimmät integrointimenetelmät ovat kokoava, jäsentävä ja big bang -lähestymistapa sekä horisontaalinen, sandwich- ja toimialakomponenttien integrointi. [4.]



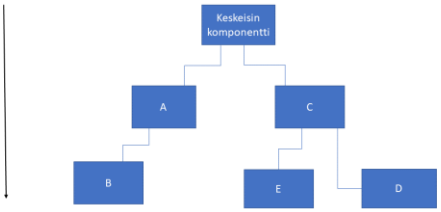
Kuva 6. Esimerkki integroitavan järjestelmän rakenteesta.

Kokoavassa lähestymistavassa, englanniksi bottom-up integration [3, luku 5.2.2] (kuva 7) hierarkiassa edetään ulkokehästä kohti koko ohjelmiston kannalta keskeisempiä komponentteja. Ensin testataan hierarkiassa alimpana oleva komponentti ja sen liittymiä hierarkiassa yläpuolella oleviin komponentteihin simuloidaan testiajureilla. Seuraavaksi siirrytään kerroksittain ylöspäin ja lopulta testataan koko ohjelmiston keskeisin komponentti. [4.]



Kuva 7. Kokoava integroinnin lähestymistapa.

Jäsentävä lähestymistapa, englanniksi top-down integration, (kuva 8) on järjestykseltään kokoavan lähestymistavan vastakohta [3, luku 5.2.1]. Jäsentävässä lähestymistavassa integroidaan ensiksi keskeisin komponentti ja siihen liittyvien komponenttien rajapintoja simuloidaan testityngillä. Testityngällä tarkoitetaan testiohjelmaa, joka antaa todellisenkaltaisen vastauksen, kun integroitava ohjelma kutsuu jotain hierarkiassa alempana olevaa rajapintaa. Jäsentävässä integroinnissa siirrytään kerroksittain hierarkiassa alaspäin, kunnes koko ohjelmisto on testattu. [4.]

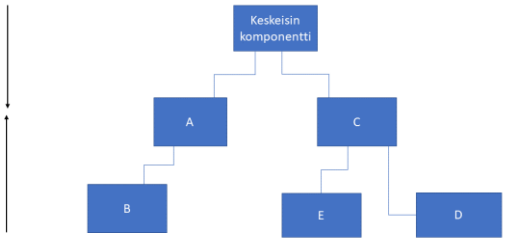


Kuva 8. Jäsentävä integroinnin lähestymistapa.

Big bang -lähestymistavassa kaikki komponentit testataan kerralla. Big bang -lähestymistapa voi säästää vaivaa, jos testejä ei tarvitse toistaa, mutta sitä käytettäessä mahdollinen vika ja sen aiheuttanut komponentti on vaikeampaa paikallistaa. Big bang -menetelmä myös viivästyttää komponenttien testaamista,

koska ei ole realistista odottaa, että kaikki komponentit valmistuvat samanaikaisesti. [3, luku 5.2.4.]

Sandwich-integrointi (kuva 9) on yhdistelmä kokoavaa ja jäsentävää lähestymistapaa. Siinä integrointi aloitetaan samanaikaisesti sekä uloimmasta että keskeisimmästä komponentista ja testataan, kunnes kaikki ohjelmiston rajapinnat on testattu. [4.]



Kuva 9. Sandwich-integrointi.

Horisontaalista integrointia käytetään, kun liitetään yhteen kokonaisia ohjelmistoja tai kun uutta ohjelmistoa liitetään olemassa olevaan tietokantaan.

Toimialakohtainen integrointi on periaatteiltaan samanlainen kuin kokoava lähestymistapa. Toimialakohtaisessa integroinnissa testausjärjestys määräytyy komponenttien välisen liikenteen perusteella. Siinä ensin määritellään joukko komponentteja, joiden välillä on useita rajapintoja, mutta joukosta ulospäin on vain muutamia. Ensin testataan joukon sisäiset rajapinnat, ja sen jälkeen valitaan seuraava komponentti tai komponenttijoukko, jolla on useita yhteisiä rajapintoja alkuperäisen komponenttijoukon kanssa ja testataan näiden ryhmien väliset rajapinnat. Tätä jatketaan, kunnes koko ohjelmisto on integroitu. [4.]

Järjestelmätestaus

Järjestelmätestauksessa koko ohjelmisto testataan ennen sen toimittamista asiakkaille. Järjestelmätestauksessa voidaan testata, täyttääkö ohjelmisto sille asetetut tavoitteet esimerkiksi käytettävyyden tai vikasietoisuuden osalta ja täyttääkö ohjelmisto sille asetetut suorituskykytavoitteet. Järjestelmätestauksen yhteydessä voidaan myös testata järjestelmän ohjeistuksia ja vastaavatko ne toteutunutta. Järjestelmätestauksessa voidaan myös tutkia, miten järjestelmä toimii kuormitettuna, esimerkiksi silloin, kun siihen syötetään suuria määriä dataa. Myös tietoturvatestaus tehdään järjestelmätestauksen osana. Järjestelmätestauksessa ohjelmistoa käytetään sellaisessa ympäristössä, kuin missä sitä on tarkoitus myös lopulta käyttää. [3, luku 6.1; 4.]

Hyväksyntätestaus

Kun toimitetaan järjestelmä asiakkaalle, asiakas yleensä tekee oman hyväksyntätestauksen, ennen kuin ottaa uuden järjestelmän käyttöönsä. Tämä vaihe voidaan asiakkaan puolella katsoa myös osaksi integraatiotestausta, kun uusi järjestelmä integroidaan osaksi vastaanottajan olemassa olevia järjestelmiä. Hyväksyntätestauksessa tutkitaan, toteuttaako toimitettu järjestelmä sille asetetut vaatimukset. [4.]

3.4 Testausympäristöt

Testausympäristö määrittyy testattavan kohteen mukaan. Testausympäristönä voi toimia yksittäisellä tietokoneella käytettävä ohjelmisto, tai testausympäristö voi olla monimutkainen useita järjestelmiä sisällään pitävä tuotantoympäristön kaltainen laitetila. Yleensä yksikkötestaus hoidetaan ensimmäisen kaltaisella, koska yksikkötesti ei vaadi kuin kyseiseen yksikköön liitettyjen rajapintojen simuloinnin. Kun tuodaan valmiiseen järjestelmään uusia osia, vaaditaan myös testaukselta enemmän. Testauksessa pitää voida selvittää, miten uusi laite tai ohjelmisto toimii olemassa olevien järjestelmien kanssa ja mitä mahdollisia ongelmia liittämisestä aiheutuu.

Puolustusvoimien yhteinen kehitys- ja integraatiotestausympäristö

Puolustusvoimien yhteinen kehitys- ja integraatiotestausympäristö eli PVITY on Puolustusvoimien järjestelmien testaukseen rakennettu ympäristö, jossa ympärivuorokautinen testaus on mahdollista. PVITY:n tavoitteena on ollut tuottaa integraatiotestausympäristö, joka tarjoaa korkealaatuista ja standardoitua testaus-, evaluointi- ja sertifiointipalvelua Puolustusvoimien järjestelmien ja niiden välisen toiminnan kehittämiseen. PVITY:ssä on asiakasprojekteille käytössä monipuoliset projektityökalut ja kattavat virtualisointipalvelut, joiden avulla voidaan tehdä tarvittavia testejä Puolustusvoimien erilaisille järjestelmille. [11.]

PVITY:n kehitys on ollut nopeaa ja jatkuvaa. Tekniikan kehittyessä myös testausympäristöä pitää kehittää, ja näin on myös toimittu PVITY:n tapauksessa. PVITY:ä on kehitetty kahden vuoden mittaisina kehitysprojekteina sekä ketterän kehityksen periaatteiden mukaan joustavasti. Pitemmillä projekteilla on tuotu käytettäväksi uusia ominaisuuksia, ja nykyisin ketterällä toiminnalla vastataan esimerkiksi kasvaneen käytön aiheuttamaan akuuttiin resurssitarpeeseen.

PVITY:ä käytetään tuotekehityksen aikana tapahtuvaan testaukseen ja muuhun vastaavaan toimintaan. PVITY:n on tuotannonkaltaisena ympäristönä tarkoitus tukea uusien tuotteiden tuotantoonsiirtoa. Tulevaisuudessa PVITY:ä on tarkoitus kehittää siten, että se tarjoaa mahdollisimman kattavasti mahdollisuudet Puolustusvoimien eri järjestelmien rajapintojen testaamiseen, jotta uusia järjestelmiä ja vanhojen järjestelmien päivityksiä voidaan testata jatkuvasti eikä vain sitä varten erikseen järjestetyissä tilaisuuksissa. [12.]

4 Haastattelut ja niiden tulokset

Insinöörityössä tehtyjen haastattelujen tavoitteena oli tuottaa tietoa siitä, miten Puolustusvoimien testausympäristöjen käyttäjät suhtautuvat tutkittaviin asioihin ja minkälaisia parannusehdotuksia heiltä saadaan nykytilanteeseen.

Haastatteluihin osallistui eri rooleissa toimivia PVITY:n käyttäjiä, kuten projektipäälliköitä, tietoturva-asiantuntijoita sekä ICT-järjestelmäpäälliköitä, jotkut useammassa roolissa.

Haastattelut tehtiin noin 30–60 minuuttia kestävinä teemahaastatteluina. Haastatteluista otettiin äänitallenne ja tehtiin kirjanpito, johon poimittiin vastauksista pääkohdat. Tämän jälkeen tehtiin taulukko jokaisesta kysymyksestä ja syötettiin sarakkeisiin saatujen vastausten avainkohdiksi tulkittavat asiat. Haastateltavien henkilöiden roolien monipuolisuuden vuoksi kaikilla haastatelluilla ei ollut kosketuspintaa ihan jokaisen kysymyksen aiheisiin, joten he vastasivat niihin, mihin heillä oli paras asiantuntemus. Seuraavaksi vertailtiin taulukossa olevia vastauksia ja koodattiin värikoodeilla useammissa haastatteluissa toistuvat, tai hyvin läheisesti toisiaan muistuttavat, vastaukset. Näistä usein toistuvista vastauksista voitiin sitten vetää johtopäätöksiä siitä, mikä on haastatelluille yleisesti tärkeä asia.

4.1 Testausympäristöt ja niiden konfiguraationhallinta

Konfiguraationhallinta on oleellinen osa testiympäristön hallinnointia. Haastattelussa kysyttiin, miten konfiguraationhallinta tulisi järjestää testausympäristössä. Useampi haastateltu kiinnitti huomiota siihen, että pitää olla mahdollista pitää tarkkaa kirjaa siitä, mitä laite- ja ohjelmistoversioita on käytössä testin aikana. Myös rajallisten resurssien osalta pitäisi olla mahdollisuus varata tarvittavat resurssit aikatauluttamalla testejä, niin että voidaan välttää testiympäristöjen käytöstä aiheutuvia konflikteja. Lisäksi ehdotettiin automatisoitua järjestelmää, josta saataisiin tallennettua kulloisenkin testin aikainen konfiguraatio.

Haastatteluissa nähtiin myös, että virtualisointi mahdollistaa erilaisten ympäristöjen testaamisen, vaikka todellista laitteistoa ei aina ole käytettävissä. Taulukkoon 1 on koottu keskeisimmät testausympäristön konfiguraationhallintaan liittyvät kommentit jokaisesta haastattelusta.

Taulukko 1. Testausympäristön konfiguraationhallinta. Toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Miten järjestää testausympäristön konfiguraationhallinta, jotta erilaiset testaustarpeet saadaan parhaiten sovitettua?				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
On kirjattava laitteet ja ohjelmistoversiot tarkasti, jotta testit ja mahdolliset viat voidaan tarvittaessa toistaa.	Aikatauluttamalla testejä varaamalla esimerkiksi kalenterista aikaa ympäristössä.	Keskitetty konfiguraationhallinta.	Virtualisoinnilla mahdollista simuloida erilaisia konfiguraatioita.	Ympäristöt jatkuvasti kehittyviä.
Olisi hyvä, jos konfiguraatiotiedot saisi järjestelmästä automaattisesti talteen.	Hajautetut ympäristöt.	Muutama henkilö vastaa konfiguraationhallinnasta.	Jos samaa laitetta tarvitaan useassa testiympäristössä, virtualisoinnilla voidaan mallintaa ja monistaa.	Pidetään tarkasti kirjaa, minkälaisessa ympäristössä testattu, jotta voidaan uusia testi tarvittaessa.
Varauskalenteri käytölle ja kapasiteetille.	Vasta lopulta yhtäaikaisia testejä.	Versiot, asetukset ja laitteet kirjattava tarkasti, jopa lämpötila ja ilmankosteus voivat vaikuttaa tuloksiin.		
Automatisointi voi auttaa välttämään virheitä.				

4.2 Testausympäristöjen laitteiston vaatimukset

Haastatteluissa kysyttiin, minkälaisia vaatimuksia laitteistolle pitäisi olla hyvässä testausympäristössä. Vastauksissa toistui eniten se, että testilaitteiston pitää olla todellisuutta vastaavaa eli sen mukaista, missä järjestelmää tullaan lopulta käyttämään. Saman verran toistui myös toivomus joustavasta konfiguraatiosta, jotta pystytään tekemään mahdollisimman monipuolista testaamista. Lisäksi toistuivat standardinmukaisuus ja päivitettävyys. Yksittäisissä vastauksissa kiinnitettiin huomiota myös tukipalveluiden toimintaan ja dokumentaation ajantasaisuuteen sekä siihen, että vikatapauksissa turvaluokiteltua tietoa sisältäviä tallennusvälineitä ei saisi lähettää valmistajan huoltoon, vaan ne olisi jätettävä Puolustusvoimille huollon ajaksi, eikä valmistajalle saa antaa etäyhteyksiä

järjestelmiin, joissa käsitellään kriittistä tietoa. Yksi huomio oli myös, että hyvä testilaitteisto ei saa aiheuttaa testeissä sellaisia virhetilanteita, joita ei voi lopullisessa käyttöympäristössä tulla. Taulukkoon 2 on koottu haastattelujen keskeisimmät kommentit liittyen testausympäristöjen laitteistovaatimuksiin.

Taulukko 2. Testausympäristön laitteiston vaatimukset. Toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Minkälaisia vaatimuksia on laitteistolle hyvässä testausympäristössä?				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
Standardinmukainen laitteisto.	Pitää olla todellisuutta vastaava laitteisto, jotta tulokset luotettavia	Monipuolista laitteistoa, että saadaan mahdollisimman paljon erilaisia testejä tehtyä.	Joustavasti konfiguroitavissa, voisi toteuttaa virtualisoinnilla.	Tuotannon kaltainen laitteisto.
Tukipalvelut pitää olla kunnossa.		Standardinmukaista.	Jotkut testit voi tarvita fyysisen laitteen.	Ei saa aiheuttaa poikkeamia, joita todellisessa ympäristössä ei voi tulla.
Päivitykset kunnossa.		Ajanmukainen laitteisto.		Skaalautuvuus, pitää voida testata myös huonon suorituskyvyn ympäristöjä.
Tallennusvälineet tiedon omistajalle huoltotapauksissa.		Päivitettävyys tärkeää.		
Ei etähallintaa valmistajalle, jos käsitellään kriittistä tietoa.		Dokumentointi ajan tasalla.		
Loppukohteen mukaista rautaa tarvittaessa.				

4.3 Testauksen automatisoinnin ja virtualisoinnin rooli

Haastattelujen mukaan testiautomaatiota kannattaa hyödyntää, koska sillä pystytään karsimaan testaajan virheitä, takaamaan yhdenmukaiset testit ja

kasvattamaan testauksen nopeutta. Eniten mainintoja tuli, että virtualisoinnilla voi saada luotua helpommin erilaisia konfiguraatioita. Kuitenkin tulee muistaa, ettei virtualisoitu testiympäristö aina vastaa todellista käyttöympäristöä. Lisäksi testaaja saattaa löytää virheitä, joita testien suunnitteluvaiheessa ei ole osattu edes miettiä. Taulukkoon 3 on koottu haastattelujen keskeisimmät kommentit liittyen virtualisointiin ja testiautomaatioon.

Taulukko 3. Testiautomaation ja virtualisoinnin rooli testausympäristössä. Toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Minkälainen rooli virtualisoinnilla ja testiautomaatiolla on nykyaikaisessa ICT-testausympäristössä?				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
Kasvava rooli, jota kannattaa hyödyntää.	Kannattaa hyödyntää uusia tekniikoita.	Voi karsia inhimillisiä virheitä.	Voidaan testata helpommin erilaisia konfiguraatioita.	PV:llä DevOps-tyyppistä toimintaa lähinnä ympäristön kehittämisessä.
Luotujen ympäristöjen asetukset nollattava testien välillä.	Saadaan testinaikaisia käyttäjävirheitä vähennettyä.	Helpommin yhdenmukaisia testejä.	Automatisoinnilla voidaan pienentää työmäärää.	
	Jopa välttämätön pitkiin testeihin, joissa selviävät esimerkiksi mahdolliset puskurien täyttymiset.	Virtualisoinnilla voi tehdä helpommin erilaisia konfiguraatioita.	Automaatiolla voidaan ajastaa päivittäin ajettavat testit matalan käyttöasteen ajaksi.	
	Voi auttaa testaamaan mahdollisimman kattavalla kalustolla.	Asiat ei aina toimi samoin virtualisoituna, kuin todellisessa maailmassa.	Voidaan tehdä automatisoituja testiskriptejä päivityksien testaamiseen.	

4.4 Jatkuva integraatio ja ketterät menetelmät

Haastatelluista suuri osa piti nykyaikaisia kehitysmenetelmiä, kuten jatkuvaa integraatiota ja ketteriä menetelmiä Puolustusvoimille haasteellisina siksi, että

heidän projekteissaan testataan lähinnä valmiita tuotteita ja myös Puolustusvoimien omat prosessit koettiin kankeiksi uusien menetelmien käyttöönottoon. Puolustusvoimien järjestelmien kehittäminen on yleensä ulkoistettu kumppaneille, ja Puolustusvoimat ostaa valmiita tuotteita ja palvelua. Tämän vuoksi toiminta on lähinnä tuotteiden hyväksyntää ja integrointia osaksi muita järjestelmiä. Puolustusvoimien yhteistyökumppanit kyllä käyttävät yhteistyöprojektien omissa osuuksissaan ketteriä menetelmiä. Mainittiin myös, että on haasteellista löytää resursseja suunnittelemaan kattava testiputki, jotta jatkuva integraatio voitaisiin ottaa käyttöön. Taulukkoon 4 on koottu haastattelujen keskeisimmät kommentit liittyen jatkuvaan integraatioon ja ketteriin menetelmiin.

Taulukko 4. Jatkuva integraatio ja ketterät menetelmät, toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Miten jatkuva integraatio (CI) ja ketterät menetelmät vaikuttavat testiympäristön suunnitteluun?				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
Koko testiputki pitää automatisoida, jos halutaan jatkuvaa kehitystä.	Ei oikein koske omia projekteja.	Sopii lähinnä ohjelmistojen testaamiseen.	PV:lle lähinnä valmista ohjelmistoa toimittajilta.	Testausympäristöä kehitetään jatkuvien periaatteiden mukaan.
Pitää sisältää tietoturvatestit.	Varmasti ajan myötä tulee enemmän käyttöön.	Uutta rautaa tuodessa haasteellista.	CI siellä yhteistyöyritysten puolella.	Yhteistyöyritykset käyttävät omassa tuotekehityksessä.
Haasteellista ulkopuolisten toimijoiden kanssa.			Voisi auttaa hyväksyntätestausta, jos perustestit saadaan ajettua automaattisesti.	Ei oikeastaan PV:n järjestelmissä käytössä, koska ostetaan valmista.
			PV:n prosessit kankeita CI:n ja ketterien menetelmien hyödyntämiseen.	

4.5 Etätyön vaikutukset testausympäristöihin

Etätyö on tuonut lisää tarvetta saada yhteys testausympäristöön myös ulkopuolelta. Etätestausta suunnitellessa kannattaa miettiä, miten mahdollisiin vikatilanteisiin puututaan. Tarvitsee olla kyky kytkeä laite pois päältä tai käynnistää se uudelleen, vaikka testaaja voisikin olla etänä. Laitteistoa testatessa fyysiset kytkennät pitää lähes aina tehdä laitteiston luona.

Yksi ratkaisu fyysisten kytkentöjen tekemiselle olisi ohjelmistolla ohjattava testiverkko, jolla voi luoda kytkentöjä eri laitteiden välille sekä poistaa niitä, mitä ei enää tarvita. Tällainen testiverkko vaatii kuitenkin, että kaikki tarvittavat portit on valmiiksi kytketty testiverkkoon, ja tämä taas vaatii varautumista useampiin valmiiksi rakennettuihin yhteyksiin kuin tilanne, jossa testaaja tai testausta tukeva organisaatio käy fyysisesti tekemässä tarvittavan kytkennän ristikytkentäkaapin luona.

Positiivisina huomioina etätyöstä nousivat käyttöasteen paraneminen, kustannussäästöt ja aikataulutuksen helpompi sovittaminen. Kustannussäästöjä syntyy erityisesti siitä, ettei Puolustusvoimien eri toimijoiden tarvitse ajaa toiselta puolelta Suomea testaamaan PVITY:n. Näin säästyy aikaa ja energiaa.

Taulukkoon 5 on koottu haastattelujen keskeisimmät kommentit etätyön vaikutuksista testausympäristössä.

Taulukko 5. Etätyön vaikutus testausympäristön suunnitteluun. Toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Miten etätyö vaikuttaa testausympäristön käyttöön, suunnitteluun ja ylläpitoon?				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
Riippuen turvaluokituksista jotain voi olla mahdollista testata etänä.	Aikataulutus voi onnistua paremmin.	Kaikkea ei voi tehdä etänä, fyysiset kytkennät voi olla ongelma.	Etätyö voi olla joustavampaa.	Voi aiheuttaa viiveitä.
Pitää olla kyky tai resurssi, jolla testilaitteen voi käydä sammuttamassa tai uudelleenkäynnistämässä.	Parempi käyttöaste mahdollinen.	Pitää olla mahdollisuus korjata etäyhteys vikatilanteissa.	Covid-pandemian aikana etätyön tekeminen on lisääntynyt.	Vikojen selvitys vaikeutuu, jos ei olla laitteen luona. Samat haasteet, kuin etätyössä muutenkin.
Parempi käyttöaste mahdollista.	Toimivat etäyhteydet voi olla ongelma.		Kustannussäästöjä ja tehokkaampaa toimintaa, kun ei ole tarvinnut matkustaa aina testilaitteen luo.	Vähentää tarvetta siirtyä paikkakunnalta toiselle, säästää aikaa testaajilta.

.

4.6 Testausympäristön tietoturva

Haastateltujen mukaan tietoturva on tärkeä osa testausympäristön suunnittelua. Usein yritysten testausympäristöissä testataan uusia, yritykselle kriittisiä versioita tuotteista, joiden päätyminen vääriin käsiin voi vaarantaa koko yrityksen toiminnan. Haastatteluissa nousivat eniten esille turvaluokitukset ja niihin suhtautuminen sekä turvaluokitusten aiheuttamat ongelmat testaukselle.

Toinen useasti huomioitu tarve oli suunnitella, miten käyttäjäoikeuksia myönnetään, ja että tarpeettomien käyttöoikeuksien poistaminen järjestelmästä on tärkeää. Myös tarve aidonkaltaiselle, mutta alhaisen turvaluokituksen testidatalle tuli useammalta haastatellulta esille, koska nykytilanteessa on jouduttu

korottamaan testien turvaluokkaa vain käytetyn testidatan takia. Turvaluokan korottaminen voi vaikeuttaa ulkopuolisten konsulttien käyttöä testaamisen apuna.

Lisäksi toivottiin tarkkaa kirjanpitoa siitä, mitä laitteistoa ja ohjelmistoa testiympäristöön tuodaan. Pitäisi myös olla tarkka loki mahdollisista hälytyksistä. Lokikirjaa pitää myös jonkun vastuullisen seurata, ei riitä, että asiat kirjataan, jos ei niille tehdä mitään.

Taulukkoon 6 on koottu haastattelujen keskeisimmät kommentit liittyen tietoturvaan.

Taulukko 6. Tietoturva testiympäristössä. Toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Mitä testausympäristön tietoturvassa pitää ottaa huomioon? Esimerkkeinä suojaukset, käyttöoikeudet ja tietoturva				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
Katakrin parhaat käytänteet. Optimistandardit, ei välttämättä aina realistinen tavoite.	Pitää olla henkilökohtaiset käyttäjäoikeudet, ei yhteiskäyttötunnuksia, jotta voidaan valvoa, kuka teki ja mitä tehtiin.	Suojaus tietoturvaluokan mukaan.	Tietoturvaluokka vaikuttaa etätestauksen mahdollisuuksiin. Pitää huomioida, että testidata voi olla korkeammalla tasolla, kuin mitä oikeastaan vaaditaan. Pitäisi osata erottaa ohjelmisto ja data toisistaan.	Riippuu testattavasta sovelluksesta. Turvaluokat iso asia, pitää huomioida testauksen suunnittelussa.
Käyttöoikeudet minimiin, vain se mitä oikeasti tarvitsee. Vanhat oikeudet pois, ei saa jäädä roikkumaan.	Ei saa olla liian tiukka käyttöpolitiikka, ettei haittaa testaamista liiaksi. Pitkien salasanojen toistuva kirjoittaminen voi kääntyä tarkoitustaan vastaan	Pidettävä kirjaa testausympäristössä olevista laitteista ja ohjelmistoista.	Tarve testidatalle, joka aidonnäköistä, mutta ei turvaluokiteltua. Voi olla haasteellista saada tarpeeksi generoitua testidataa kuormitustesteihin.	Aina ei voi käyttää todellista dataa. Simuloidun testidatan suunnitteluun ja tekemiseen pitää varata aikaa.
Tarkka kirjanpito mitä ympäristöön on tuotu		Käyttöloki pitää olla. Se voi olla automaattinen tai manuaalinen.		
Laaja lokikirja, jota jonkun pitää myös seurata.		Käyttäjäkohtaiset tunnukset, geneerisiä ei saisi olla kuin erityississä poikkeustapauksissa.		
Tietoturvan taso pitää määritellä testien mukaan.				

Käyttöoikeudet

Käyttöoikeuksista testausympäristöön haastatteluissa vastattiin, että oikeuksia pitää valvoa. Oikeudet pitää myöntää vain tarvittaviin osiin testausympäristöä,

eikä niitä pidä antaa vain varmuuden vuoksi. Vastauksissa suositeltiin myös, että käyttöoikeuksia pitää antaa vain tarvittavalle ajalle, ettei jää oikeuksia roikkumaan henkilöille, jotka eivät niitä enää käytä. Suositeltiin myös, että jokaisella on oma käyttäjätunnus eikä käytetä mitään yhteistunnuksia kuin erittäin harvoin poikkeustapauksissa. Omien tunnusten käyttö helpottaa käytön seurantaa. Haastatteluissa huomautettiin kuitenkin, että käyttöperiaatteen pitää olla sen verran löysä, ettei se haittaa testaamista liiaksi. Esimerkiksi pitkien vaikeiden salasanojen syöttäminen useita kertoja käsin testatessa voi kääntyä tarkoitustaan vastaan. Tähän voi olla ratkaisuna testiympäristön erottaminen muista järjestelmistä toistuvien testien ajaksi, jolloin tämän salasanan vaatimuksilla ei olisi ihan niin paljon merkitystä. Haastatteluissa tuli myös ilmi, että pääkäyttäjän salasanat pitää säilyttää turvallisessa paikassa, ne eivät voi olla vain yhden henkilön tiedossa. Pääkäyttäjän salasanat pitää myös vaihtaa määräajoin.

Testiympäristön valvonta

Testausympäristön valvonta on tärkeä osa testiympäristön tietoturvaa. Haastatteluissa tuli useasti esiin lokikirjan pitäminen erilaisista tapahtumista testausympäristössä. Pitää myös varata resurssit lokien seuraamiseen, koska lokikirjalla ei tee mitään, jos ei sitä kukaan seuraa eikä hälytyksiin reagoida. Pitää myös tallentaa tarkasti, mitä laitteistoa tai ohjelmistoja on tuotu testausympäristöön, jotta voidaan jälkikäteen jäljittää mahdollisten ongelmien syitä.

Tietoturvaohjelmistot

Mahdolliset tietoturva- ja virustorjuntaohjelmistot pitäisi testausympäristössä olla sen mukaiset, mitä ne lopullisessa käyttöympäristössä ovat, jottei tule yllättäviä tilanteita testauksessa tai lopullisessa käytössä. Laitteista, joista ei ole pääsyä ulkopuolelle, voidaan tarvittaessa poistaa tietoturvaohjelmistot käytöstä testien ajaksi.

Ulkopuoliset yhteydet

Tietoturvan taso pitää määritellä testien mukaan. Lähtökohtaisesti yhteyksien ulkomaailmaan testattavasta laitteistosta tulisi olla haastattelujen perusteella mahdollisimman rajattuja, mieluummin niitä ei pitäisi olla ollenkaan. Jos on pakko kytkeä jotain, niin silloin vain yksisuuntaisia tai mahdollisesti välityspalvelimen kautta olevia erittäin rajattuja yhteyksiä, jotta testausympäristöä voidaan suojella mahdollisilta ulkopuolisilta hyökkäyksiltä.

4.7 Usean käyttäjän testausympäristön hallinta

Kun on useita käyttäjiä ja rajalliset resurssit, niistä tulee usein kilpailua. On syytä suunnitella, miten näitä rajallisia resursseja jaetaan. Resurssi voi testausympäristön tapauksessa tarkoittaa esimerkiksi fyysistä tilaa, laskentakapasiteettia, yksittäistä laitetta, verkkoliikennettä tai jotain muuta rajallista asiaa, jota voi testauksessa tarvita. Haastatteluissa tähän esitettiin ratkaisuna varauskalenteria tai vastaavaa järjestelmää, jonka ylläpitäjä ja projektit yhteistyössä tarvittaessa jakavat testausresurssit tarvitsijoille.

Etätyö, virtualisointi ja automatisointi ovat helpottaneet resurssipulaa, koska automatisoinnin avulla voi testejä tarvittaessa ajoittaa ruuhka-aikojen ulkopuolelle eikä fyysinen tila tai laitekaan enää virtualisoinnin myötä ole välttämättömyys. Esimerkiksi pitkäkestoisia, paljon laskentaa vaativia kuormitustestejä voitaisiin mahdollisesti suunnitella automatisoitaviksi vapaapäivien ajalle. Taulukkoon 7 on koottu haastattelujen keskeisimpiä kommentteja liittyen testausympäristön hallintaan.

Taulukko 7. Testausympäristön hallinta. Toistuvat teemat koodattuna väreillä, yksittäiset valkoisella pohjalla.

Miten voi toteuttaa usean käyttäjän testausympäristön hallinnan? Esimerkkinä testien aikatauluttaminen.				
Haastattelu1	Haastattelu2	Haastattelu3	Haastattelu4	Haastattelu5
Varauskalenteri on hyvä olla resursseille	Eri aikavälit määritelty erilaisille testeille. Ajastaminen.	Resurssien varausjärjestelmä	Ohjelmistolla ohjattavat yhteydet, ei jää "roikkumaan" tarpeettomia yhteyksiä	Virtualisoinnilla voidaan saada enemmän resursseja tarvitsijoille
Virtualisointi voi mahdollistaa useamman testauksen samanaikaisesti		Dokumentaatio pitää olla ajan tasalla, jotta näkee, mitä voi varata.		Konfliktia resurssien saamisessa tulee välttää. Jonkun pitää jakaa resurssit.
Kuormittavien testien ajastaminen matalan käyttöasteen ajalle		Konflikteissa pitää olla joku, joka päättää resurssien käytön tärkeysjärjestyksen. Ylläpito määrittelee käytettävät resurssit		

4.8 Haastattelujen johtopäätökset

Haastatellut asiantuntijat haluavat hyvän standardeihin tukeutuvan ympäristön, joka voi tarjota monenlaisia testauskonfiguraatioita. Haastatelluille oli tärkeää pitää huolta tietoturvasta, ja huolenaiheena oli resurssien riittävyys. Ratkaisuksi resurssipulaan nähtiin virtualisointi ja testiautomaatio, joita on jo PVITY:ssä laajasti käytössä. Myös etätyö nähtiin mahdollisena vastauksena olemassa olevien resurssien tehokkaampaan käyttöön. Yksi selkeimmistä puutteista oli laadukkaan aidonkaltaisen testidatan tarve, joka on joissain tapauksissa johtanut tietoturvaluokan tarpeettomantuntuiseen nostamiseen, koska joudutaan käyttämään aitoa, testattavaa kohdetta korkeamman turvaluokituksen omaavaa dataa. Lisäksi esille nousi resurssivarausjärjestelmän kehittäminen ja varsinkin siitä tiedottamisen tarpeellisuus.

Toimenpide-ehdotuksia

Haastatteluiden pohjalta insinöörityössä esitettiin seuraavia toimenpide-ehdotuksia:

Toimenpide-ehdotus 1, laitteistovaatimukset

- Kannattaa myös jatkossa pysyä standardien mukaisessa laitteistossa, jolloin laitteiston hankinta esimerkiksi vikaantuneiden tilalle voidaan toteuttaa useammalta mahdolliselta toimittajalta.

Toimenpide-ehdotus 2, konfiguraationhallinta

- Järjestelmävastuullisen on selvitettävä, pystytäänkö tuottamaan järjestelmä, josta testaajat voivat saada automaattisesti tulostetun testikonfiguraation.

Toimenpide-ehdotus 3, automatisointi ja virtualisointi

- Testiautomaatiota kannattaa kehittää, ja varsinkin toistuvat perustestit kannattaisi automatisoida.

Toimenpide-ehdotus 4, ketterät menetelmät ja jatkuva kehitys

- Ketteryyttä kannattaa lisätä ja kehittää prosesseja, jotta uusien versioiden käyttöönottoa saadaan nopeutettua.

Toimenpide-ehdotus 5, etätyö

- Testausympäristöön tarvitaan kontaktihenkilö, joka voi tarvittaessa auttaa ongelmatilanteiden ratkomisessa etätyöskentelijöiden tukena.

Toimenpide-ehdotukset 6 ja 7, ympäristön hallinta

- Resurssien varauskäytännöistä tiedottaminen on testausympäristön kannalta merkityksellistä.
- Testausympäristön tueksi tulisi kehittää priorisointijärjestelmä, jolla on selkeä vastuuhenkilö. Tämän henkilön toimivaltuuksiin kuuluu päätöksenteko resurssien jakamisesta, jos tarvitsijoita on enemmän kuin kapasiteettia.

Toimenpide-ehdotus 8, tietoturva

- Testausympäristön toimijoiden hyödynnettäväksi tulisi kehittää aidonkaltaista testidataa tai jokin järjestelmä, jolla sellaista voidaan nopeasti luoda testitapauksia varten. Lisäksi kehitetään ympäristön lokikirja, jonka avulla voidaan jälkikäteen tietää, mitä ympäristössä on tapahtunut.

5 Yhteenveto

Insinöörityössä tehtävänä oli tutkia Puolustusvoimille ICT-testausympäristön järjestelyitä. Tutkimusmenetelmänä käytettiin teemahaastattelua, ja haastatteluihin osallistui viisi Puolustusvoimien yhteisen kehitys- ja integraatiotestausympäristön käyttäjäprojektien avainhenkilöä.

Työtä tehdessä huomattiin, että testausympäristö on käsitteenä hyvin laaja ja se voi tarkoittaa jotain aina yksittäisestä testiohjelmasta monimutkaiseen tuotannonkaltaiseen testiympäristöön. Puolustusvoimissa ei itse tehdä tuotekehitystä, vaan testauksessa on kyse valmiiden tuotteiden integroinnista olemassa olevaan järjestelmään, joten testausympäristönä on useimmin laaja ympäristö.

Todettiin, että kun tehdään monimutkaisen järjestelmän testausta, standardit helpottavat testien tulosten vertailua ja antavat myös selkeyttä laitteiden hankintaan. Myös tietoturva-asioissa kannattaa noudattaa kansallisen turvallisuusviranomaisen Katakrin eli kansallisen turvallisuusauditointikriteeristön ohjeita.

Haastatteluissa todettiin myös, että nykyaikaisten menetelmien, kuten ketteryys, automatisointi, virtualisointi ja etätyö, käyttö on lisääntynyt myös Puolustusvoimissa ja niillä nähtiin olevan pääasiassa positiivisia vaikutuksia, vaikka myös jotain haittoja nähtiin varsinkin etätyön osalta.

Tutkimuksen tuloksena tuotettiin toimenpide-ehdotuksia, joita voidaan käyttää Puolustusvoimissa testausympäristöjen kehittämisen lähdeaineistona.

Lähteet

1 Hirsjärvi, Sirkka; Remes, Pirkko & Sajavaara Paula. 1997. Tutki ja kirjoita. Helsinki: Kirjayhtymä.

2 Eskola, Jari & Suoranta, Juha. 1998. Johdatus laadulliseen tutkimukseen. Tampere: Vastapaino.

3 Srinivasan, Desikan & Gopalaswamy, Ramesh. 2007. Software Testing Principles. E-kirja. Pearson India.

4 Räsänen, Seppo. Ohjelmiston testaus ja laatu. Verkkoaineisto. Savonia AMK. <http://webd.savonia.fi/home/ktrasse/muut/testaus_laatu/testaus_3.pdf>. Luettu 24.1.2022.

5 Ketterän menetelmän ohje. 2022. Puolustusvoimien logistiikkalaitos. Luettu 28.4.2022.

6 Schwaber, Ken. 2009. Agile Project Management with Scrum. E-kirja. Microsoft Press.

7 Kim, Gene; Humble, Jez; Debois, Patrick & Willis, John. 2016. The DevOps Handbook. IT Revolution.

8 Piikkila, Jessica. What is SAFe. Verkkoaineisto. <https://www.atlassian.com/agile/agile-at-scale/what-is-safe>. Luettu 28.4.2022.

9 Harle, Raija. SAFe tulee ja pelastaa. Verkkoaineisto. <https://www.cgi.com/fi/fi/blogi/safe-tulee-ja-pelastaa>. Luettu 28.4.2022.

10 Scaled Agile. Verkkoaineisto. <https://www.scaledagile.com>. Luettu 28.4.2022.

11 Projektisuunnitelma Puolustusvoimien integraatiotestausympäristö. 2015. Järjestelmäkeskus.

12 PVITY:n kotisivut. Puolustusvoimien logistiikkalaitos. Luettu 28.4.2022