Verkkopalvelun palvelimien valvontaohjelman kehitys

Joona Korhonen

2022 Laurea

Laurea-ammattikorkeakoulu

Verkkopalvelun palvelimien valvontaohjelman kehitys

Joona Korhonen

Tietojenkäsittely

Opinnäytetyö

Toukokuu, 2022

Laurea-ammattikorkeakoulu **Tiivistelmä**

Tietojenkäsittely

Tradenomi (AMK)

Joona Korhonen

Verkkopalvelun palvelimien valvontaohjelman kehitys

Vuosi 2022 Sivumäärä 31

Opinnäytetyössä suunniteltiin ja toteutettiin palvelimien valvontaohjelma toimeksiantajana toimivalle Visma Real Estate Oy:lle. Toimeksiantajalla ei ollut aikaisemmin käytössä valvonta-ohjelmaa palvelimille. Valvontaohjelman oli tarkoitus olla mahdollisimman yksinkertainen sekä varmatoiminen ja ohjelman asetuksia piti pystyä muuttamaan erillisestä asetustiedos-tosta. Valvontaohjelman tavoitteena oli helpottaa virhetilanteiden havaitsemista sekä no-peuttaa virhetilanteisiin reagointia.

Ohjelma kehitetiin PowerShell- ja Python-ohjelmointikielillä. Kehitetty ohjelma hyödynsi useita rajapintoja. Kehityksessä käytettiin vesiputousmallia. Valittu vaihejakomalli sopi hyvin tähän kehitysprojektiin, koska kyseessä oli pieni ja yksinkertainen ohjelma, jonka vaatimukset toimeksiantaja pystyi määrittelemään tarkasti ennen projektin aloitusta. Valvontaohjelmisto täytti määritellyt vaatimukset ja valmistui aikataulussa. Toimeksiantaja oli tyytyväinen loppu-tulokseen.

Asiasanat: ohjelmistokehitys, rajapinta, python, powershell, palvelinylläpito

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's thesis

Joona Korhonen

Development of web services servers monitoring software

Year 2022 Pages 31

The purpose of this thesis was to design and develop a server monitoring software for Visma Real Estate Oy which commissioned the development project. Visma Real Estate Oy did not have monitoring software for servers. Monitoring software was supposed to be simple and robust. The software had to have separate configuration file for software's configurations. The main goals for the monitoring software were to make it easier to notice if serious error had happened on a server and enable faster reaction time to serious errors.

Software was developed using PowerShell and Python programming languages. The developed software utilized multiple application interfaces. Development of the software was made with waterfall model. The chosen software development life cycle model suited well for this project, because the project aim was to make small and simple software. The commissioner was able to define the requirements clearly before the software development was started. The result of the project fulfilled the commissioner's requirements, expectations and was completed on schedule. The commissioner was satisfied with the result.

Keywords: software development, application interface, python, powershell

Sisällys

1 Johdanto

Toiminnallisessa opinnäytetyössä tehtävä valvontajärjestelmän osa on yhdistelmä palvelimen sisäänrakennettuja ominaisuuksia ja näitä hyödyntävää rajapintaa. Työssä tutustutaan aiheeseen liittyviin palvelimen ominaisuuksiin sekä miten niitä voidaan ohjelmallisesti käyttää osana valvontaohjelmaa. Käsitellään ja selvennetään ohjelmistokehitykseen sekä ohjelmiston toimintaan liittyviä keskeisiä käsitteittä, työvaiheita sekä järjestelmän vaatimuksia.

Toteutettavassa ohjelmassa käytetään datan hakemisessa hyödyksi useita eri rajapintoja. Lukijalle pyritään antamaan kattava kokonaiskuva siitä miten monimuotoisia rajapinnat ovat ja minkälaisia ominaisuuksia niiden avulla voidaan ohjelmaan toteuttaa.

Teoriaosuudessa käydään läpi ohjelmistokehitysprojektin vaiheita ja tutustutaan yleisimpiin ohjelmistokehitysprojektin vaihejakomalleihin.

2 Toimeksiantaja

Toimeksiantajana tässä opinnäytetyössä toimii Tampuuri -nimistä kiinteistönhallintaohjelmistoa tuottava Visma Real Estate Oy. Toimeksiantajan tuottamaa ohjelmistoa tarjotaan SaaS-palveluna, eli ohjelmistoa ei tarvitse asentaa laitteelle, vaan ohjelmistoa käytetään internetin välityksellä.

2.1 Visma Real Estate Oy

Visma Real Estate Oy on osa Visma-konsernia. Visma-konserniin kuuluu yli 200 yritystä 20 maasta. Visma on viiden merkittävimmän ohjelmistoyrityksen joukossa EU:n alueella. (Visma Tampuuri, 2021.)

Visma Real Estate Oy:llä on kaksi tuotetta; Hausvise ja Tampuuri. Yrityksessä työskentelee noin 140 henkilöä (Visma Tampuuri, 2022). Hausvise on isännöintijärjestelmä, joka yhdistää toiminnanohjauksen, taloushallinnon ja viestinnän. Tampuuri on kiinteistönhallintajärjestelmä, joka sisältää yli 50 tuotemoduulia sekä integraatiota muiden palveluiden välillä.

3 Työn lähtökohdat

3.1 Kehittämiskohteen kuvaus ja kehittämistavoitteet

Opinnäytetyössä tehtävän kehitystyön aloite tuli toimeksiantajan tarpeesta palvella asiakasta paremmin. Opinnäytetyössä käsitellään SaaS-palveluna tarjottavan Tampuuri ohjelmiston tuotantoympäristön palvelimien valvontaohjelman kehitystä ja toteutusta. Työn tavoitteena on parantaa nykyistä verkkopalvelun tuotantoympäristön valvontaa vastaamaan paremmin yrityksen tarpeita. Yksinkertaistetaan nykyistä virheviestien sisältöä sekä mietitään uudelleen virheviestin lähetykseen käytetyt viestintäkanavat.

Valvontajärjestelmä kokonaisuudessaan jakautuu kolmeen eri kerrokseen: verkko-, palvelin- ja sovelluskerrokseen. Verkkokerroksessa valvontaan kuuluu sisäverkon yhteyden sekä palvelimien yhteyden ulkoverkkoon. Palvelinkerroksessa valvotaan prosessorin kuormitusta, keskusmuistin käyttöä, levyjen käyttöä ja palvelimen komponenttien tilaa. Sovelluskerroksessa käsitellään ja välitetään Tampuuri -sovelluksessa tapahtuneita virheitä.

Valmis valvontajärjestelmä tulee koostumaan kolmesta erillisestä valvontaohjelmasta. Jokainen valvontaohjelma tallentaa virheet myös tietokantaan, virheviestin lähettämisen lisäksi.

3.2 Aihealueen rajaus

Opinnäytetyössä määritellään valvottavat toiminnot, tehdään valvontaohjelma ja perustetaan virheviestien välityskanava. Opinnäytetyössä tehdään valvontajärjestelmän palvelimia valvova osa, mutta työstä ulkopuolelle jää virheiden tallentaminen tietokantaan, käyttöliittymän tekeminen virheiden esittämiseen, virheiden automaattinen korjaus sekä valvontajärjestelmän verkko- ja sovelluskerrokset. Työn ulkopuolelle jäävät osat kehitetään myöhemmin.

3.3 Keskeiset käsitteet

IDRAC	IDRAC (Intergrated Dell Remote Access Controllers) on IPMI-standardiin perustuva käyttöliittymä, joka hyödyntää BMC-ohjainta. IDRAC mahdollistaa palvelimen hallinnan etänä. (Kennedy, 2018.)
API	Application programming interface (API) tarkoittaa ohjelmointirajapintaa. Ohjelmointirajapinnat mahdollistavat tiedonvälityksen eri palveluiden ja ohjelmien välillä. (IBM Cloud Education, 2020.)
JSON	JavaScript Object Notation (JSON) on tiedonvälitykseen käytetty yksinkertainen tiedostomuoto. Suurin osa ohjelmointikielistä tukee JSON-tiedostomuotoa. (W3Schools, 2022.)
Webhook	Tiedonvälitysmenetelmä, jonka avulla voidaan luoda omia http-kutsuja. Kutsut voidaan suorittaa esimerkiksi määrätyn tapahtuman yhteydessä. (SendGrid Team, 2014.)
Python	Python on tulkattava korkeantason ohjelmointikieli (Python, 2022).
PowerShell	PowerShell on korkeantason ohjelmointikieli, jota käytetään pääasiassa järjestelmien ylläpidon automatisointiin (Microsoft, 2022).
SaaS	Software as a Service on ohjelmiston jakelumalli, jossa palvelintarjoaja ylläpitää sovellusta palvelimillaan ja sovellus välitetään asiakkaille internetin välityksellä. (Oracle, 2020.)
Postman	Postman on API-ohjelmointirajapintojen testausta varten kehitetty ohjelmisto (Postman, 2022).
IP	Internet Protocol (IP) on osoite, jonka avulla laitteesi tunnistetaan verkossa. IP-osoite on yksilöllinen. (Kaspersky, 2022.)

4 Ohjelmistokehitys

Ohjelmistokehitys on prosessi, missä tuotetaan vaatimusmäärittely, analysoidaan vaatimukset, suunnitellaan ratkaisu ja tuotetaan tietokoneohjelma. Ohjelmistokehitys ei kuitenkaan ole sama kuin ohjelmistosuunnittelu. Ohjelmistokehitys on osa ohjelmistosuunnittelua, mutta ohjelmistosuunnittelu pitää sisällään myös prosessinhallintaa ja muita toimintoja. Ohjelmistokehitys on siis ohjelmistosuunnittelun tietokoneohjelman tuottamiseen keskittynyt osa. (Dooley 2011, 1.)

4.1 Ohjelmiston elinkaari

Ohjelmiston kehityksen alussa täytyy valita ohjelmistolle vaihejakomalli. Ohjelmiston elinkaari koostuu eri vaiheista, joissain ohjelmistoissa vaiheita voidaan yhdistää, mutta kaikki ohjelmiston elinkaaret suorittavat nämä vaiheet: (Dooley 2011, 7.)

1. Esitutkimus
2. Määrittely
3. Suunnittelu
4. Toteutus
5. Testaus
6. Julkaisu
7. Ylläpito
8. Ohjelmiston elinkaaren loppu

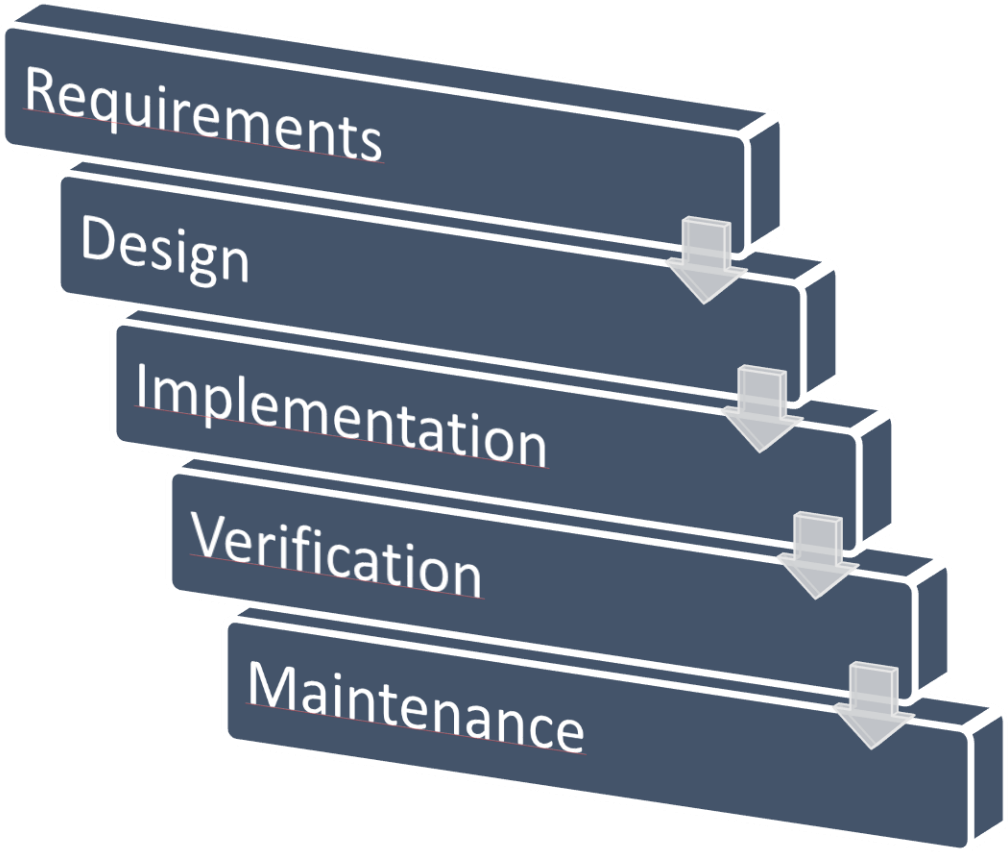
(Dooley 2011, 7.)

Kaikilla ohjelmilla on elinkaari, mutta ne voi suorittaa elinkaaren vaiheet eri tavalla. Elinkaaret voidaan jakaa kahteen pääryhmään. Ensimmäisessä ryhmässä käydään kaikki vaiheet läpi, ennen kuin siirrytään takaisin elinkaaren ensimmäiseen vaiheeseen, tai ainakin suoritetaan vaiheet kahdesta seitsemään. Toisessa ryhmässä käydään vaiheet osittain läpi, yleensä vaiheet kolmesta viiteen ennen kuin edetään julkaisuvaiheeseen. (Dooley 2011, 7.)

Ohjelmiston vaihejakomalleja on useita erilaisia. Muutamia suosituimpia on waterfall-, agile-, spiral- ja V-malli. (Martin, 2022.)

4.1.1 Vesiputousmalli

Vesiputousmalli eli waterfall-model on ensimmäinen ohjelmistokehityksessä käyttöönotettu ohjelmiston vaihejakomalli. Vesiputousmallin käyttö aloitettiin 1970-luvulla. (Hughey, 2009.) Vaikka nykyäänkin vesiputousmalli on suosittu, on sen käyttö vähentynyt nykyaikaisempien mallien kehityksen myötä. (W3schools, 2022.)



Kuvio 1. Vesiputousmalli

Kuvion 1 mallissa esitutkimus ja määrittely, sekä testaus- ja julkaisuvaiheet on yhdistetty.

Vesiputousmallissa oletetaan, että kaikki ohjelmiston vaatimukset saadaan kerättyä mallin ensimmäisessä vaiheessa. (Hughey, 2009.)

Suunnitteluvaihe voidaan jakaa kahteen alavaiheeseen; loogiseen ja fyysiseen suunnitteluvaiheeseen. Loogisessa suunnitteluvaiheessa suunnitellaan ohjelman toimintalogiikka ylätasolla, tässä vaiheessa ei vielä kohdejärjestelmää oteta huomioon. Fyysisessä suunnitteluvaiheessa aloitetaan loogisen suunnitelman soveltaminen kohdejärjestelmälle. (Hughey, 2009.)

Implementointivaiheessa ohjelmoijat tuottavat ohjelmankoodin suunnitelmien mukaisesti. (Hughey, 2009.)

Kun ohjelmankoodi on kirjoitettu, siirrytään varmistusvaiheeseen. Varmistusvaiheessa tarkastetaan, että ohjelmisto vastaa mallin ensimmäisessä vaiheessa määriteltyjä vaatimuksia ja myös suoritetaan ohjelmiston testaus mahdollisten ohjelmistovirheiden varalta.
Tässä vaiheessa ohjelmiston käyttö aloitetaan. (Hughey, 2009.)

Mallin viimeisessä vaiheessa ohjelmisto on asiakkaan käytössä. Ohjelmistoon voidaan tehdä muutoksia ja korjauksia virheellisten määrittelyiden tai muiden virheiden vuoksi.
(Hughey, 2009.)

Vesiputousmallin etuja:

- Mallin ansiosta ohjelmiston valmistumista on helppoa seurata, koska mallissa edetään vaiheittain
- Jokaisessa vaiheesta tuotetaan hyvä dokumentointi seuraavaa vaihetta varten.

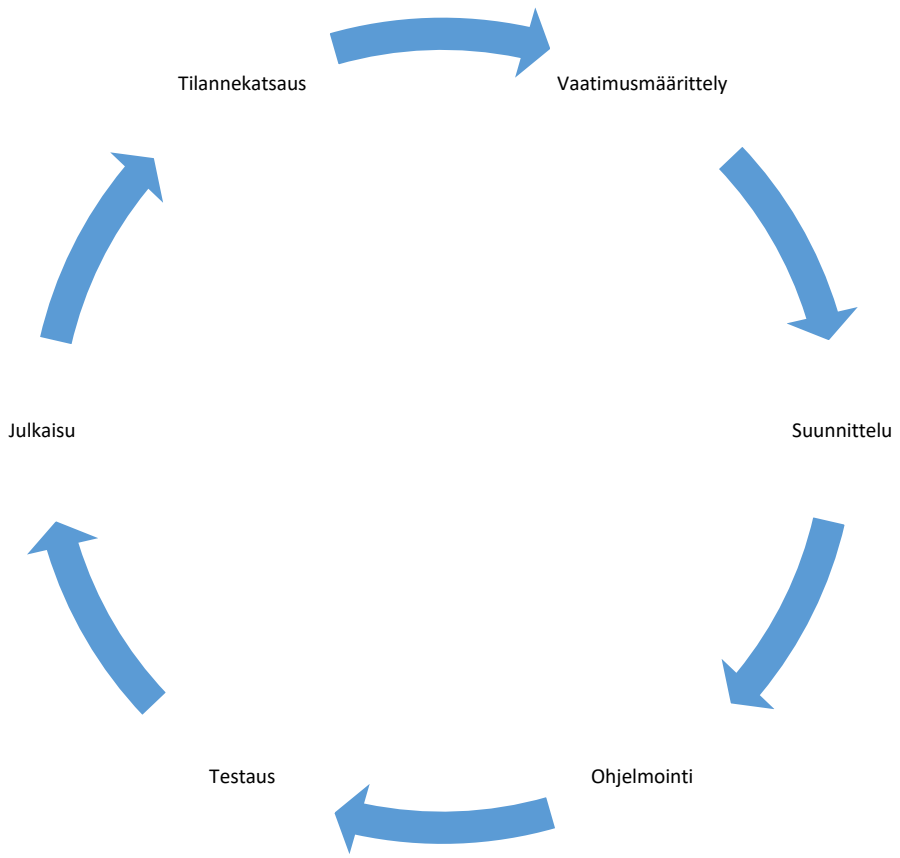
Vesiputousmallin heikkoudet:

- Ohjelmiston toiminnallisuudet pitää pystyä määrittämään ensimmäisessä vaiheessa ja yleensä puutteet määrittelyssä huomataan vasta kun ohjelmisto on toimitettu.
- Vaatimusten muuttaminen ei ole mahdollista implementointivaiheessa.

Vesiputousmalli soveltuu paremmin pienempiin ohjelmointiprojekteihin, missä ohjelmiston toiminnallisuudet pystytään määrittelemään etukäteen tarkasti. (Martin, 2022.)

4.1.2 Ketterä -malli

Ketterän ohjelmistokehitysmallin tarkoitus on tehdä ohjelmistokehityksestä joustavaa, nopeampaa ja jatkuvaa (Doshi, 2019). Ketterä -kehitysmalli on sateenvarjotermi viitekehyksille ja toimintatavoille, mitkä perustuvat ketterä -mallin manifestiin (Agile Alliance, 2021) sekä noudattaa mallin 12:ta periaatetta. (Agile Alliance, 2022)



Kuvio 2. Ketterä -malli

Ketterä ohjelmistokehitysmalli koostuu iteraatioista, missä vaiheet suoritetaan yhä uudelleen ja uudelleen (Kuvio 2). Jokainen iteraatio kehittää ohjelmistoa palautteen mukaan tehtävillä muutoksilla. (Yu, Wooi & Wai, 2012.)

Ketteräkehityksen menetelmiä on useita, mutta niistä suosituin on Scrum (Diogo, 2022). Scrum tunnetaan kehitysjaksoista, joita kutsutaan sprinteiksi. Sprintti kestää yhdestä neljään viikkoa, jonka aikana tuotetaan määritelmän mukainen julkaisukelpoinen ohjelma. Scrumissa työ tehdään pienissä tiimeissä, joita kutsutaan Scrum -tiimeiksi. Scrum -tiimi kuuluu Scrum Master, tuoteomistaja ja kehittäjiä. Scrum -tiimistä löytyy kaikki osaaminen mitä sprintillä tarvitaan. (Schwaber & Sutherland, 2020.)

Tiimit ovat itseohjautuvia ja päättävät tiimin kesken kuka tekee, mitä tekee ja miten tekee. Tiimien on tarkoitus pysyä pieninä, että kommunikaatio pysyy helppona ja tiimi ketteränä. Jos tiimi uhkaa kasvaa liian isoksi, voidaan tiimi jakaa kahdeksi tiimiksi, joilla on samat tavoitteet ja tuoteomistaja. Tuoteomistaja, vastaa ohjelmiston arvon ja tiimin työn arvon maksimoimisesta. Tuoteomistaja määrittelee tuotteen vaatimukset ja järjestää kehitysjonon

yhdessä tiimin kanssa. Scrum Master pitää huolen siitä, että kaikki ymmärtää scrum periaatteet teoriassa ja käytännössä. Scrum Master pitää huolen siitä, että tiimin työskentely on tuottavaa ja että tiimi noudattaa scrum -käytänteitä. (Schwaber & Sutherland, 2020.)

Scrumin tuotoksia hallitaan tuotteen kehitysjonon ja sprintin tehtävälistan kautta ja jokaisen sprintin tuloksena on valmis tuoteversio. Vaikka sprintin tuotoksena saadaan valmis tuoteversio, ei sitä välttämättä julkaista ja tämän päätöksen tekee tuoteomistaja. Scrum -tapahtumilla luodaan säännöllisyyttä ja vähennetään tarvetta muille kuin scrum -palavereille. (Schwaber & Sutherland, 2020.)

Scrum -tapahtumat:

- Sprintti
- Sprintin suunnittelu
- Päivittäinen palaveri
- Sprintin katselmus
- Sprintin retrospektiivi

(Schwaber & Sutherland, 2020.)

Kaikki Scrum tapahtumat on aikarajattuja, esimerkiksi sprintin suunnittelupalaveri voi kestää maksimissaan 8 tuntia, jos sprintin pituus on kuukauden. Päivittäiset palaverit taas kestävät 15 minuuttia. (Schwaber & Sutherland, 2020.)

Sprintin aikana tuotetaan määritelmän mukainen julkaisukelpoinen ohjelma. Sprintit ovat saman pituisia koko kehityksen ajan. Uusi sprintti alkaa heti edellisen päätyttyä. Kaikki työ mukaan lukien sprintin suunnittelu, päivittäiset palaverit, sprintin katselmus ja sprintin retrospektiivi, tapahtuu sprinttien sisällä. (Schwaber & Sutherland, 2020.)

Sprintin suunnittelu on sprintin ensimmäinen tapahtuma ja tässä määritetään sprintillä tehtävät työt. Päivittäisillä palavereilla seurataan sprintin tavoitteiden saavuttamista. Sprintin katselmuksessa on tarkoitus tarkastella sprintin tuotosta, sekä scrum -tiimi esittelee tuotoksen sidosryhmille. Scrum -tiimi yhdessä sidosryhmien kanssa selvittää mitä sprintin aikana saavutettiin ja mietitään seuraavat kehityskohteet. (Schwaber & Sutherland, 2020.)

Sprintin retrospektiivissä käydään läpi, miten sprintti sujui työn tekemisen näkökulmasta. Käydään tiimin kanssa läpi sprintin onnistumiset ja mahdolliset ongelmat, jos ongelmia esiintyi, miten ne ratkaistiin. (Schwaber & Sutherland, 2020.)

4.2 Esitutkimus ja vaatimusmäärittely

Esitutkimusvaiheessa pyritään selvittämään asiakkaan tarpeet sekä selvitetään syy miksi ohjelmisto tulisi kehittää (Pöyhönen, 2005).

Vaatimusmäärittelyssä määritellään ohjelman toiminnallisuudet ja näkymät käyttäjän näkökulmasta. Vaatimusmäärittelyä voisi pitää huonosti tehtynä käyttöohjeena sovellukselle, koska tässä vaiheessa ei vielä tiedetä yksityiskohtaisesti, miten toiminto tai näkymä toimii. (Dooley 2011, 38.)

Vaatimukset jaetaan kahteen ryhmään; toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnallisilla vaatimuksilla tarkoitetaan ohjelmiston toimintoja. Ei-toiminalliset vaatimukset ovat laadullisia vaatimuksia, esimerkiksi toimintaympäristön asettamat rajoitteet tai tietoturvaan liittyvät vaatimukset. (Luukkainen, 2021.)

Ketterässä ohjelmistokehityksessä vaatimusmäärittelyn työkaluista tärkein on User Story eli käyttäjätarina. Käyttäjätarinat kirjoitetaan asiakkaan ymmärrettäväksi, koska käyttäjätarinan avulla on tarkoitus kuvata käyttäjän kannalta, miten toiminnallisuus tuottaa arvoa. (Luukkainen, 2021.)

Käytettävästä vaihejakomallista riippumatta vaatimusmäärittelyn tulee alkaa ennen ohjelmiston suunnittelua ja toteutusta. Vaatimusmäärittely tehdään yhdessä asiakkaan tai loppukäyttäjän kanssa. (Luukkainen, 2021.)

4.3 Suunnittelu

Suunnitteluvaiheessa käydään läpi vaatimusmäärittelyssä olevat vaatimukset. Suunnitteluvaiheessa suunnitellaan miltä ohjelma tulee näyttämään, miten ohjelma käyttäytyy ja miten ohjelma käsittelee ohjelmaan syötettyä tai ohjelman käytettävissä olevaa dataa. Tässä vaiheessa myös päätetään se, millä ohjelmointikielellä ohjelma tullaan toteuttamaan.

Ennen ohjelmiston toteutusvaihetta voidaan myös tehdä suunnitelman pohjalta prototyyppi ja siitä saatujen tulosten perusteella tehdä muutoksia alkuperäiseen suunnitelmaan. (Aristek System, 2021.)

4.4 Ohjelmiston toteutus

Ohjelmiston toteutusvaiheessa kehittäjät alkavat kirjoittamaan ohjelman lähdekoodia valitulla ohjelmointikielellä. Toteutusvaiheen tehtävät jaetaan pienempiin osiin tai moduuleihin, joita sitten voidaan jakaa kehittäjille tai tiimeille. Kehittäjän on noudatettava ohjelmointikielen ja yleisesti ohjelmointiin liittyviä ohjesääntöjä toteutuksessa. (Martin, 2022.)

4.5 Testaus

Kun toteutusvaihe on valmis, siirrytään testausvaiheeseen. Testausvaiheessa tarkastetaan, että vaatimusmäärittelyssä määritellyt ohjelmiston toiminnot toimivat vaatimusten mukaisesti (Kasurinen, 10). Testauksen suorittaa yleensä ohjelmistotestaajista koostuva testaustiimi (Martin, 2022).

4.6 Ylläpito

Ohjelmiston ylläpitovaihe alkaa, kun ohjelmisto on läpäissyt testausvaiheen ja se on julkaistu käyttäjän saataville ja käyttäjät alkavat käyttämään ohjelmistoa (Javatpoint, 2022). Ohjelmiston ylläpitoon kuuluu:

- Ohjelmistovirheiden korjaaminen
- Ohjelmiston päivittäminen
- Ohjelmiston ominaisuuksien jatkokehitys.

(Martin, 2022.)

Ohjelmistoon voidaan tuoda esimerkiksi uusia tietoturvaominaisuuksia ohjelman julkaisun jälkeen ja näin pidentää ohjelmiston käyttöikää. Ohjelmiston toimintoja voidaan päivittää, että ohjelmiston käyttö olisi sujuvampaa. Kaikkia ohjelmistovirheitä ei huomata testausvaiheessa ja käytön aikana havaitut virheet korjataankin ylläpitovaiheessa. (Martin, 2022.)

5 Kehitystyön kohde

Toimeksiantajan Tampuuri tuote toimii toimeksiantajan palvelinsaleissa olevilla palvelimilta sekä osittain pilvipalvelussa olevilla virtuaalipalvelimilta. Suurin osa ohjelmiston toiminnoista toimii toimeksiantajan omilla palvelimilla. Pilvipalvelussa on ainoastaan joitain osia tuotteesta, esimerkiksi ohjelman kirjautumisesta vastaava moduuli on pilvipalvelussa.

Toimeksiantajan palvelinsaliin tulee ostettuna palveluna ainoastaan internetyhteys, kaikki muu on toimeksiantajan itse ylläpidettävänä. Palvelimien ongelmatilanteista saatavat virheilmoitukset ovat erittäin tärkeitä, koska ohjelmisto toimii verkkoselaimella verkon yli ja palvelimet ovat toimeksiantajan omassa ylläpidossa. Toimeksiantajalla on fyysisiä palvelimia noin 20 kappaletta ja näillä palvelimilla on eri tehtäviä, esimerkiksi tietokantapalvelimia, aineistopalvelimia ja virtuaalikoneiden isäntäpalvelimia.

Ohjelmiston verkkosivut löytyvät aina kolmelta palvelimelta ja nämä palvelimet ovat kuormantasaajan takana. Varotoimista huolimatta, jos fyysisen palvelimen komponentti vikaantuu, täytyy siitä saada ilmoitus mahdollisimman pian, että palvelimelle voidaan varata

huolto ja ongelmatilanne saadaan ratkaistua. Kehitystyön kohteena on siis kehittää ohjelma, jolla saadaan palvelimen virheestä ilmoitus nopeasti ja varmasti ylläpitotiimin tietoon.

6 Kehittämishankkeen menetelmät ja toteutus

Opinnäytetyön kehittämishanke suoritettiin projektina ja ohjelmiston vaihejakomallina käytettiin lineaarista vesiputousmallia, koska kehittämishankkeen tarkoituksena oli tuottaa pieni ohjelma, jonka toiminnot tiedettiin hyvin jo etukäteen.

6.1 Nykytilan kartoitus

Nykytilan kartoitus aloitettiin tarkastelemalla tuotantoympäristön palvelimia. Ensimmäisenä todettiin, että kaikki tuotantoympäristön palvelimet olivat samalta valmistajalta. Palvelimien ominaisuuksiin tutustuessa, havaittiin palvelimissa olevan etähallinta ominaisuus. Etähallinta ominaisuutta varten palvelimissa on oma verkkokortti ja prosessori varattuna vain palvelimen etähallintaan. Etähallintaominaisuuden käyttöliittymää voi käyttää selaimen kautta. Palvelimien etähallintaominaisuus on nimeltään iDrac ja se on Dellin versio IPMI-standardiin perustuvasta käyttöliittymästä.

Etähallintaominaisuutta voi myös käyttää Redfish-standardiin perustuvan rajapinnan välityksellä, jonka kautta voidaan hakea tietoa palvelimen komponenttien tilasta sekä muuttaa palvelimen asetuksia. Jokaisella palvelimella on etähallintaa varten oma IP-osoite ja hallintapaneeli, mutta palvelimille on myös yhteinen etähallintapaneeli. Palvelimien yhteisestä etähallintapaneelista on mahdollista käyttää toimintoa mikä lähettää sähköposti-viestin, kun jollain siihen liitetyllä palvelimella tapahtuu virhe.

Virheestä saadaan viesti, mutta ongelmaksi muodostuu se, että viesti virheestä menee yhteiskäyttösähköpostiin ja sen seuraaminen työllistää tarpeettomasti ylläpitotiimiä. Virheviestit voi myös helposti kadota sähköpostilaatikkoon, koska samaan sähköpostilaatik-koon tulee virheviestejä useista eri lähteistä ja osa näistä virheistä ei ole kriittisiä.

6.2 Tavoitteiden määritys

Tavoitteena on tehdä ohjelma, joka tarkastaa palvelimien komponenttien tilan ja lähettää virheviestin, mikäli palvelimen komponentissa on havaittu vika. Virheviestit pitää olla helposti havaittavissa sekä selkeitä. Ohjelmaan pitää myös pystyä helposti lisäämään valvottavia palvelimia ilman, että ohjelman lähdekoodiin tarvitsee tehdä muutoksia. Tietojen haun epäonnistuminen yhdeltä ei saa kaataa koko ohjelmaa, vaan poikkeukset täytyy käsitellä ohjelman sisällä.

6.3 Tiedonhankinta

Aloitin työn tiedonhankinnan tutustumalla yrityksen palvelininfran dokumentaatioihin. Selvitin, että yrityksellä on kaksi palvelinsalia, toinen Salossa ja toinen Helsingissä. Yrityksellä oli käytössä kolme toimialuetta. Tässä työssä viittaamme näihin toimialueisiin käyttämällä aakkosten kolmea ensimmäistä kirjainta eli a, b ja c. Näistä a-toimialue oli varattu VPN-yhteydelle ja tästä toimialueesta on pääsy b- ja c-toimialueisiin, joista ei muuten ollut auki ulkoverkkoon kuin muutama portti, mutta ei etäyhteysmahdollisuutta. A-toimialueesta saa siis yhteyden kaikkiin valvonnanpiiriin kuuluviin palvelimiin.

Käyttöjärjestelmänä palvelimissa oli Windows Serverin eri versioita sekä muutamassa virtuaalipalvelimessa Linuxin eri versioita. Kaikissa toimialueen ohjauspalvelimissa on käyttöjärjestelmänä Windows.

Työssä käytettiin pääosin julkisesti saatavilla olevia rajapintoja ja moduuleita. Tästä syystä suurin osa tarvittavasta tiedosta löytyi ohjelmisto- tai laitevalmistajilta. Vikasietoisuutta lisätäkseni käytössäni oli myös yrityksen sisäiseen käyttöön tarkoitettu sähköpostin lähetyksen mahdollistava rajapinta.

Palvelimien valmistajalla on saatavilla hyvät dokumentit palvelimen etähallintarajapinnasta ja sen toiminnasta (Dell, 2022). Rajapinnan dokumentoinnista löytyy mitä tietoja on haettavissa rajapinnan kautta ja mistä osoitteesta tieto löytyy. Tiedonhankintavaiheessa rajanpinnan palauttamaan dataan pääsi tutustumaan Postman-ohjelmalla.

Viestin välityksen mahdollisuuksia selvittäessä päädyin käyttämään Slack-viestintäsovellusta, koska sen käyttö oli juuri aloitettu yrityksessä ja sovelluksessa sai helposti organisoitua viestit eri kanaville. Slackissä löytyi myös kaksi eri tapaa välittää viestejä, webhook tai API-rajapinnan kautta. Slackissä webhook asetetaan tietylle kanavalle ja ohjelma muodostaa osoitteen mihin viestinlähetyskutsut välitetään. Käyttäessä API-rajapintaa voidaan lähettää viestejä, vaikka yksittäisille käyttäjille tai useaan eri kanavaan. Webhookit on sidottu yhteen kanavaan, eikä kutsun mukana tarvitse välittää tietoa mihin viesti lähetetään, mutta API-rajapintaa käyttäessä täytyy kutsun mukana antaa tieto mihin viesti halutaan välittää.

Toinen vaihtoehto oli Telegram-viestintäsovellus, mutta sen käyttö koettiin hieman vaikeaksi, koska Telegram ei ollut muuten käytössä yrityksessä ja Telegram oli käyttöliittymältä sekavampi.

Tiedonhakua ohjelmointikielten osalta tehtiin yrityksessä aikaisemmin käytössä olleiden ohjelmien kautta. Tiimin aikaisemmin käyttämiä kieliä oli PowerShell ja Python. Molemmista kielistä on olemassa todella paljon dokumentaatiota sekä aktiiviset yhteisöt, joista on helppoa saada apua vaikeampiinkin ongelmiin. Molemmat ohjelmointikielet ovat erittäin suosittuja tämän kaltaisissa pienissä ja yksinkertaisissa ohjelmissa.

6.4 Kehittämishankkeen toteutus

Tiedonhaun perusteella kehittämishanke päätettiin toteuttaa PowerShell ja Python-kieliä käyttämällä. Palvelimilta tiedot haetaan Redfish-rajapinnasta ja viestin välitykseen käytetään Slack-ohjelman webhookia. Ohjelman asetuksille tehtiin oma XML-muotoinen asetustiedosto. Ohjelman käyttämät salasanat tallennettiin salattuihin tiedostoihin. Ennen ohjelman varsinaista käyttöönottoa, ohjelmaan määriteltiin viestin lähetys päälle, joka ilmoitti viestillä ohjelman suoriutumisesta. Myöhemmin kun todettiin sekä viestin lähetyksen että ohjelman suorituksen toimivan, pystyttiin poistamaan testauksessa käytetty viestin lähetys.

7 Tulokset

Työn tuloksena tuotettiin ohjelma, joka kyselee palvelimelta rajapinnan kautta palvelimen komponenttien ja moduulien statustiedot.

Valvontaohjelma koostuu useasta osasta ja moduulista. Osat on toteutettu eri ohjelmointikielillä. Osiossa käytetty ohjelmointikieli on valittu ohjelmanosan käyttötarkoituksen mukaan. Käynnistysosa toteutettiin PowerShell-ohjelmointikielellä, koska sillä on tarkoitus lukea Windows palvelimella olevia tiedostoja. Pääohjelma taas toteutettiin Python-ohjelmointikielellä, koska useat käyttöjärjestelmät saadaan tukemaan Python-kielellä toteutettuja ohjelmia. Ratkaisuun päädyttiin, koska tulevaisuudessa jos halutaan suorittaa pääohjelmaa esimerkiksi Linux-palvelimelta, ei koko ohjelmaa tarvitse toteuttaa uudestaan, vaan ainoastaan tapa millä pääohjelma käynnistetään eli käynnistysosa.

- Käynnistysosa (PowerShell)
- Pääohjelma (Python)
    - Tilakyselymoduuli (Python)
    - Viestinlähetysmoduuli (Python)

7.1 Valvontaohjelma

Valvontaohjelman kehittäminen aloitettiin tekemällä suppea vaatimusmäärittely. Vaatimusmäärittelyssä määriteltiin:

- Mitä valvotaan
- Miten valvotaan
- Miten virheestä ilmoitetaan
- Ilmoitusviiveestä
- Käyttäytyminen virhetilanteessa

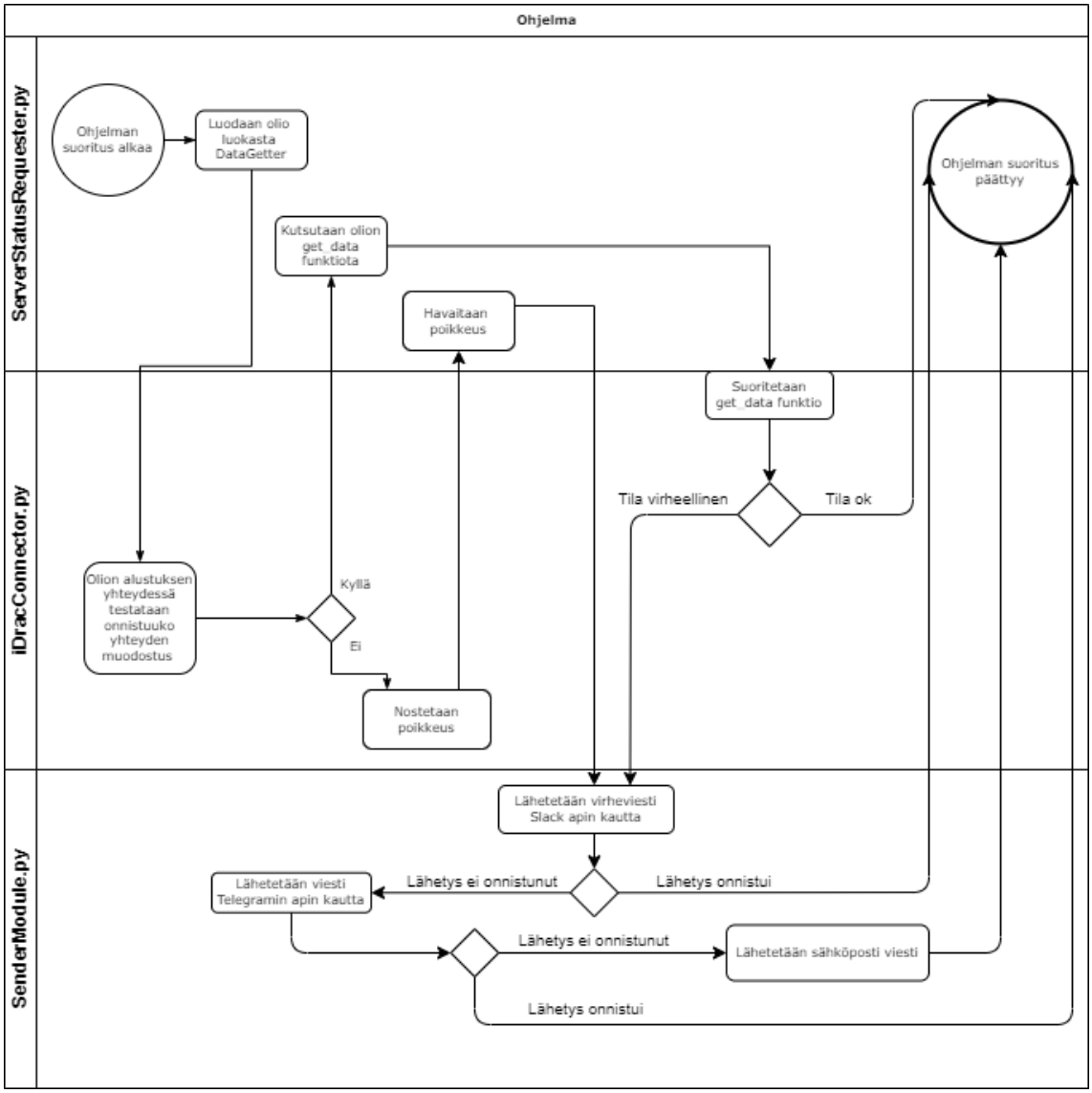
Määrittelyssä todettiin, että valvotaan fyysisiä palvelimia etähallintarajapinnan kautta ja valvottavien palvelimien listaa pitää päästä muuttamaan erillisestä asetustiedostosta.

Ohjelman piti myös olla tarpeeksi selkeä, että kuka tahansa tiimin jäsenistä voi sitä tarvittaessa muokata. Tästä syystä ohjelma toteutetaan PowerShell- ja Python-kielillä, koska niitä on käytetty tiimissä aikaisemminkin tämän tyyppisissä toteutuksissa.

Virheviestit lähetetään ensisijaisesti käyttäen Slack-ohjelman rajapintaa ja jos viestin lähetys epäonnistuu, lähetetään viesti Telegram-ohjelman rajapinnan kautta. Slackiin tehdään oma kanava jokaiselle eri virhetyypille, mutta tässä työssä käytettään ainoastaan palvelimen komponenttivirheille tarkoitettua kanavaa.

Todettiin, että virheen esiintymisestä ilmoituksen saamiseen 30 minuuttia on riittävä aikaväli.

Määrittelyssä todettiin, että ohjelmalla on mahdollisuus kahteen kriittiseen virheeseen; kyselyn epäonnistuminen ja virheviestin lähetys. Kyselyn epäonnistumisen varalta ohjelmaan tehdään virheenkäsittely, joka lähettää viestin epäonnistuneesta kyselystä. Viestin lähetyksen epäonnistuessa täytyy ohjelmassa olla useampi eri viestintäkanava, että yhden vikaantuminen ei estä viestintää täysin.



Kuvio 3. Ohjelman vuokaavio

7.2 Käyttöympäristö

Palvelinympäristö koostuu kolmesta toimialueesta; a, b ja c. Valvottavat palvelimet ovat b- ja c-toimialueessa. Palvelimien etähallintarajapinnan kyselyt tehdään palvelimien etähallinta-verkkokortille asetettuihin sisäverkon IP-osoitteisiin. Toimialueiden palvelimien etähallintara-japintaan pääsee kiinni sisäverkon IP-osoitteella toimialueen ohjauskoneelta. Ohjelma täytyy asettaa suoriutumaan kaikille toimialueen ohjauskoneille, joissa ohjelmaa halutaan suorittaa.

7.3 Ohjelman suorittaminen

```powershell
<#
        .SYNOPSIS
        Käynnistää palvelimen status kysely ohjelman
        .DESCRIPTION
        Lukee asetus-tiedostosta sinne määriteltyjen palvelimien osoitteet ja hakee etähallinan salasanan .sec-tiedostosta.
        Käynnistää palvelimen status kysely ohjelman ip osoitteella ja salasanalla
        .EXAMPLE
        Invoke-AddADUser -userName $userName -domain $domain -password $pwd [-Verbose];
#>

#Otetaan kansio mistä scripti suoritetaan muuttujaan
$kansio = Split-Path -Parent $MyInvocation.MyCommand.Path

#Haetaan Settings.xml tiedosto scriptin kansiosta
[xml]$asetukset = Get-Content $kansio"\Settings.xml"

#Loopataan Settings.xml tiedostossa olevat IP-osoitteet läpi
$asetukset.Settings.Servers.Ip.split(";") | foreach {
    #Otetaan IP-osoite muuttujaan
    $ip = $_

    #Rakennetaan polku .sec tiedoston luo mihin salasana on salattuna tallennettu
    $splitted = $_.split(".")
    $salasanaSec = $($kansio+"\"+$splitted[$($splitted.length -1)]+".sec")

    #Avataan salasana käytettävään muotoon
    $avain = (9,4,2,46,32,34,146,222,1,5,80,5,42,53,33,233,1,243,2,7,6,5,35,43);
    $salasana = cat $salasanaSec | convertto-securestring -key $avain;

    #Käynnistetään status kysely ohjelma
    python D:\Temp\Joona\oppari\ServerStatusRequester.py -ip $ip -u root -p $salasana
}
```

Kuvio 4. Käynnistysohjelma

Ohjelmaan tehtiin oma käynnistysohjelma ja tämä käynnistysohjelma toteutettiin PowerShellilla. Käynnistysohjelman tehtävänä on lukea ohjelman asetukset ulkoisesta xml-muodossa olevasta asetustiedostosta. Asetuksissa on määritelty valvottavien palvelimien IP-osoitteet, rajapinnan käyttäjätunnus sekä salasana. Toimeksiantajalla rajapinnan salasana on jokaisella palvelimella yksilöllinen.

Salasanat on tallennettu salattuun sec-tiedostoon, tiedostojen nimeämislogiikka käyttää palvelimen IP-osoitteen viimeistä kahdeksanbittistä lukua tiedoston nimenä. Tällä tiedostojen nimeämislogiikalla on helppo hakea oikea tiedosto. Nimeämislogiikka ei muodosta nimi ristiriitoja, koska palvelimien käytössä olevat osoitteet ovat 192.168.100.0/24 eli C-luokka. Tiedoston salaus täytyy purkaa ennen kuin sen sisältämä tieto voidaan välittää eteenpäin seuraavalle ohjelman osalle. Salasanatiedoston salaus puretaan käyttämällä lähdekoodissa näkyvää avainta (Kuvio 4).

Käynnistysohjelma käynnistää pääohjelman ja välittää argumentteina pääohjelmalle palvelimen IP-osoitteen, käyttäjätunnuksen sekä salasanan. Käynnistysohjelma käy läpi kaikki asetustiedostossa olevat palvelinten IP-osoitteet toistolausekkeella.

Käynnistysohjelma ajastettiin Windows-käyttöjärjestelmän ”ajoitetut tehtävät” -toiminnallisuudella, b- ja c-toimialueiden ohjauskoneille. Ohjelma on ajastettu suoriutumaan joka 30 minuutin välein, koska 30 minuutin suoritusväli tuntui sopivalta. Vaikka palvelimen komponenteissa esiintyvät virheet ovat erittäin kriittisiä, niin noin vuorokauden reaktioaika esimerkiksi kovalevyn rikkoutumiseen on riittävä. Tärkeintä on, että levy saadaan vaihdettua muutaman päivän sisään ongelman ilmenemisestä.

```
import SenderModule as sm, argparse, warnings, idracConnector as ic

warnings.filterwarnings("ignore")

parser=argparse.ArgumentParser(description="Python script using Redfish API DMTF to get health check results and overall information about the server")
parser.add_argument('-ip',help='iDRAC IP address', required=True)
parser.add_argument('-u', help='iDRAC username', required=True)
parser.add_argument('-p', help='iDRAC password', required=True)

args=vars(parser.parse_args())

idrac_ip=args["ip"]
idrac_username=args["u"]
idrac_password=args["p"]

if __name__ == "__main__":
    try:
        connector = ic.DataGetter(idrac_ip, idrac_username, idrac_password)
        connector.get_data()
    except Exception as e:
        print(e)
        sender = sm.MessageSender(idrac_ip)
        sender.send_message("Status Check Failed!")
```

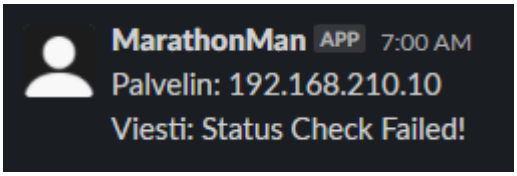
Kuvio 5. Pääohjelma

Käynnistysohjelman suorittama Pythonilla toteutettu ohjelma koostuu useasta moduulista; pääohjelmasta, kyselymoduulista ja viestinlähetysmoduulista. Kun pääohjelma käynnistetään, luo pääohjelma DataGetter -luokasta olion. Oliolle asetetaan IP-osoite, käyttäjänimi sekä salasana.

```
import requests, json, sys, re, time, SenderModule as sm
from requests.models import Response

class DataGetter:

    def __init__(self, ip, user, pwd):
        self.ip = ip
        self.user = user
        self.pwd = pwd
        try:
            response = requests.get('https://%s/redfish/v1/Systems/System.Embedded.1' % (ip),verify=False,auth=(user, pwd))
            self.data = response.json()
            self.computerSystemVersion = self.data[u'@odata.type']
            self.systemStatusDict = {}
            self.sender = sm.MessageSender(self.data[u'HostName'])
        except:
           raise ConnectionError("Yhteyden muodostaminen ei onnistunut")
```

Kuvio 6. DataGetter -luokan konstruktori

Luokalle on määritelty konstruktori, mikä suoritetaan aina kun luokasta tehdään olio. Konstruktorissa asetetaan IP-osoite, käyttäjätunnus ja salasana olion muuttujiin, sekä testataan, onnistuuko yhteyden muodostus määriteltyyn osoitteeseen. Jos yhteyden muodostus ei onnistu, nostetaan tästä yhteysvirhepoikkeus. Nostettu poikkeus käsitellään pääohjelmassa.



Kuvio 7. Epäonnistuneen yhteydenmuodostuksen virheviesti

Virhe käsitellään pääohjelmassa ja siitä lähetään viesti virhekanavalle, josta se on helposti nähtävissä (Kuvio 7). Kun käyttäjä huomaa viestin, voi hän alkaa tutkimaan tarkemmin mistä syystä yhteydenmuodostus ei ole onnistunut.

Kuvio 8. Kyselyn vastaus PowerShell-konsolissa

Jos ohjelman suorittaa PowerShell-konsolista ohjelma tulostaa ruudulle kaikki sen kyselyjen kautta saamat tiedot (Kuvio 8). Tästä on hyötyä, mikäli ohjelman toiminnassa esiintyy ongelmia tai jos käyttäjä haluaa tarkastella dataa itse.

7.4 Rajapintakysely

```
def get_data(self):
    self.get_rollup_info()
    self.get_sys_info()
    self.get_fan_info()
    self.get_psu_info()
    self.get_disk_info()
    systemStatusJSON = json.dumps(self.systemStatusDict)
    print(systemStatusJSON)
    #self.sender.send_message("Status Check Done!")
```

Kuvio 9. get_data -funktio

Rajapintakyselyt suorittava moduuli on toteutettu Python-ohjelmointikielellä ja se suoritetaan pääohjelmassa. Kyselyt suoritetaan kutsumalla olion get_data -funktiota. Funktio yhdistää kaikki DataGetter -luokan kyselyfunktiot.

Kyselymoduuli kyselee rajapinnan kautta ja rajapinta palauttaa pyydetyt tiedot JSON-muodossa. Kyselyt on jaoteltu omiin funktioihin, joita sitten kutsutaan get_data -funktiossa (kuvio 9). Funktioiden sisällä vastaanotettu data puretaan ja tallennetaan avainarvolistaan.

```
def get_rollup_info(self):
    rollupStatusDict = {}
    rollupStatusDict["Battery"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'BatteryRollupStatus']
    rollupStatusDict["Cpu"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'CPURollupStatus']
    rollupStatusDict["Fan"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'BatteryRollupStatus']
    rollupStatusDict["Intrusion"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'FanRollupStatus']
    rollupStatusDict["Licensing"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'IntrusionRollupStatus']
    rollupStatusDict["PS"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'PSRollupStatus']
    rollupStatusDict["Storage"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'StorageRollupStatus']
    rollupStatusDict["SysMem"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'SysMemPrimaryStatus']
    rollupStatusDict["Temp"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'TempRollupStatus']
    rollupStatusDict["Volt"] = self.data[u'Oem'][u'Dell'][u'DellSystem'][u'VoltRollupStatus']
    self.systemStatusDict["Rollup"] = rollupStatusDict
    for e in rollupStatusDict:
        if rollupStatusDict[e].lower() != "ok":
            self.sender.send_message("%s feilaa." %(e))
```

Kuvio 10. get_rollup_info

```
def  get_sys_info(self):
    basicStatusDict = {}
    basicStatusDict["Palvelin"] = self.data[u'HostName']
    basicStatusDict["Malli"] = self.data[u'Model']
    basicStatusDict["Prosessori"] = {self.data[u'ProcessorSummary'][u'Model']:self.data[u'ProcessorSummary'][u'Status'][u'Health']}
    basicStatusDict["Keskusmuisti"] = {self.data[u'MemorySummary'][u'TotalSystemMemoryGiB']:self.data[u'MemorySummary'][u'Status'][u'Health']}
    self.systemStatusDict["Basics"] = basicStatusDict
    for e in basicStatusDict["Keskusmuisti"]:
        if basicStatusDict["Keskusmuisti"][e].lower() != "ok":
            self.sender.send_message("%s - Tyyppi: %s." %("Keskusmuisti",basicStatusDict["Keskusmuisti"][e]))
    for e in basicStatusDict["Prosessori"]:
        if basicStatusDict["Prosessori"][e].lower() != "ok":
            self.sender.send_message("%s - Tyyppi: %s." %("Keskusmuisti",basicStatusDict["Prosessori"][e]))
```

Kuvio 11. get_sys_info

Funktiot get_rollup_info ja get_sys_info eivät tee uusia rajapintakyselyjä, vaan hyödyntävät konstruktorin suorittaman rajapintakyselyn saamaa dataa. Funktiossa get_rollup_info käsiteltävä data koostuu palvelimen suorittaman diagnostiikkatestin tuloksista (Kuvio 10.). Funktiossa get_sys_info data koostuu palvelimen perustiedoista, joita on palvelimen nimi, malli, prosessori ja keskusmuistin määrä (Kuvio 11).

```
def  get_fan_info(self):
    fanStatusDict = {}
    fanNum = 1
    for i in self.data[u'Links'][u'CooledBy']:
        singleFan = i[u'@odata.id']
        responseFan = requests.get('https://%s%s' % (self.ip,singleFan),verify=False,auth=(self.user, self.pwd))
        #print(singleFan)
        if responseFan.status_code == 200:
            fanData = responseFan.json()
            if self.computerSystemVersion == "#ComputerSystem.v1_5_0.ComputerSystem":
                #print("\t********%s********\n\tStatus: %s \n\tPyörimisnopeus: %s\n" % (fanData[u'FanName'], fanData[u'Status'][u'Health'], fanData[u'Reading']))
                fanStatusDict[fanData[u'FanName']] = fanData[u'Status'][u'Health']
            elif self.computerSystemVersion == "#ComputerSystem.v1_8_0.ComputerSystem":
                for fan in fanData[u'Fans']:
                        if fan[u'Name'].find(str(fanNum)) > 0:
                            #print("\t********%s*******\n\tStatus: %s\n\tPyörimisnopeus: %s" % (fan[u'Name'], fan[u'Status'][u'Health'], fan[u'Reading']))
                            fanStatusDict[fan[u'Name']] = fan[u'Status'][u'Health']
                        fanNum+=1
            elif self.computerSystemVersion == "#ComputerSystem.v1_10_0.ComputerSystem":
                for fan in fanData[u'Fans']:
                        if fan[u'Name'].find(str(fanNum)) > 0:
                            #print("\t********%s*******\n\tStatus: %s\n\tPyörimisnopeus: %s" % (fan[u'Name'], fan[u'Status'][u'Health'], fan[u'Reading']))
                            fanStatusDict[fan[u'Name']] = fan[u'Status'][u'Health']
                        fanNum+=1
            else:
                print("\tVersiolle %s ei ole käsittelyä" %(self.computerSystemVersion))
        else:
            print("Nyt taitaa olla joku pielessä!!!")

    self.systemStatusDict["Fans"] = fanStatusDict
    for e in fanStatusDict:
        if fanStatusDict[e].lower() != "ok":
            self.sender.send_message("%s feilaa." %(e))
```

Kuvio 12. get_fan_info

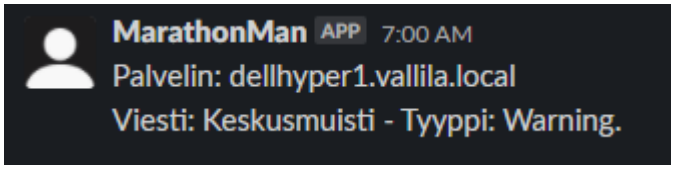
```
def get_psu_info(self):
    psuStatusDict = {}
    psuNum = 1
    for i in self.data[u'Links'][u'PoweredBy']:
        singlePsu = i[u'@odata.id']
        responsePsu = requests.get('https://%s%s' % (self.ip,singlePsu),verify=False,auth=(self.user, self.pwd))
        if responsePsu.status_code == 200:
            psuData = responsePsu.json()
            #print(singlePsu)
            if self.computerSystemVersion == "#ComputerSystem.v1_5_0.ComputerSystem":
                #print("\t**********"+psuData[u'MemberId']+"**********\n\tSn: '+psuData[u'SerialNumber']+'\n\tPn: '+psuData[u'SparePartNumber']+'\n\tStatus: '+psuData[u'Status'][u'Health']+'\n')
                psuStatusDict[psuData[u'MemberId']] = psuData[u'Status'][u'Health']
            elif self.computerSystemVersion == "#ComputerSystem.v1_8_0.ComputerSystem":
                responsePsu = requests.get('https://%s/redfish/v1/Dell/Chassis/System.Embedded.1/Power/PowerSupplies/DellPowerSupply/PSU.Slot.%s' % (self.ip,psuNum),verify=False,auth=(self.user, self.pwd))
                psuData = responsePsu.json()
                #print("\t**********%s**********\n\tStatus: %s" % (psuData[u'Id'],psuData[u'OperationalStatus'][0]))
                psuStatusDict[psuData[u'Id']] = psuData[u'OperationalStatus'][0]
                psuNum += 1
            elif self.computerSystemVersion == "#ComputerSystem.v1_10_0.ComputerSystem":
                responsePsu = requests.get('https://%s/redfish/v1/Dell/Chassis/System.Embedded.1/Power/PowerSupplies/DellPowerSupply/PSU.Slot.%s' % (self.ip,psuNum),verify=False,auth=(self.user, self.pwd))
                psuData = responsePsu.json()
                #print("\t**********%s**********\n\tStatus: %s" % (psuData[u'Id'],psuData[u'OperationalStatus'][0]))
                psuStatusDict[psuData[u'Id']] = psuData[u'OperationalStatus'][0]
                psuNum += 1
            else:
                print("\tVersiolle %s ei ole käsittelyä" % (self.computerSystemVersion))
        else:
            print("Nyt taitaa olla joku pielessä!!!")

    self.systemStatusDict["Powersupplies"] = psuStatusDict

    for e in psuStatusDict:
        if psuStatusDict[e].lower() != "ok":
            self.sender.send_message("%s feilaa." %(e))
```

Kuvio 13. get_psu_info

```
def get_disk_info(self):
    controllerStatusDict = {}
    diskStatusDict = {}
    storage = self.data[u'Storage'][u'@odata.id']
    responseStorage = requests.get('https://%s%s' % (self.ip,storage),verify=False,auth=(self.user, self.pwd))
    if responseStorage.status_code == 200:
        storageData = responseStorage.json()
        for i in storageData[u'Members']:
            print(i[u'@odata.id'])
            responseController = requests.get('https://%s%s' % (self.ip,i[u'@odata.id']),verify=False,auth=(self.user, self.pwd))
            controllerData = responseController.json()
            if controllerData[u'Status'][u'Health']:
                print("\n\t\t"+controllerData[u'Description']+"\n\t\t\tStatus: "+controllerData[u'Status'][u'Health']+"\n\t_____\n")
                controllerStatusDict[controllerData[u'Description']] = controllerData[u'Status'][u'Health']
            if controllerData[u'Drives'] != []:
                for drive in controllerData[u'Drives']:
                    responseDrive = requests.get('https://%s%s' % (self.ip,drive[u'@odata.id']),verify=False,auth=(self.user, self.pwd))
                    driveData = responseDrive.json()
                    print("\t***** "+driveData[u'Manufacturer']+" - "+driveData[u'Model']+" *****\n\t"+"Status "+driveData[u'Status'][u'Health']+"\n\t"+driveData[u'Description']+"\n")
                    diskStatusDict[driveData[u'Description']] = driveData[u'Status'][u'Health']
        for e in controllerStatusDict:
            if controllerStatusDict[e].lower() != "ok":
                self.sender.send_message("%s feilaa." %(e))

        for e in diskStatusDict:
            if diskStatusDict[e].lower() != "ok":
                self.sender.send_message("%s feilaa." %(e))


        self.systemStatusDict["DiskController"] = controllerStatusDict
        self.systemStatusDict["Disks"] = diskStatusDict

    else:
        print("Nyt taitaa olla joku pielessä!!!")
```

Kuvio 14. get_disk_info

Funktiot get_fan_info (Kuvio 12), get_psu_info (Kuvio 13) ja get_disk_info (Kuvio 14) suorittavat uudet rajapintakyselyt alkuperäisestä datasta saataviin osoitteisiin. Ensimmäisellä funktion sisällä suoritettavalla rajapintakyselyllä saadaan moduulien tai komponenttien määrä. Funktio suorittaa vielä toisenkin rajapintakyselyn millä saadaan kaikkien moduulien tai komponenttien tiedot. Funktio myös tarkastaa rajapinnan version ja suorittaa sitten rajapintakyselyn kyseiselle versiolle sopivasta osoitteesta.



Kuvio 15. Virheviesti

Funktioiden lopussa käydään toistolausekkeella läpi listalla olevat arvot. Jos tilatieto poikkeaa vertailuarvosta, kutsutaan viestinlähetysmoduulia ja moduulille välitetään

palvelimen nimi, virheellisen arvon palauttaneen komponentin tai moduulin nimen sekä virheen vakavuuden, jos tieto on saatavilla (kuvio 15).

7.5 Viestinlähetys

```
import requests
from slack_sdk import WebClient
from slack_sdk.errors import SlackApiError
from requests.models import Response

class MessageSender:

    def __init__(self, where):
        self.where = where

    def send_message(self, message):
        client = WebClient("xxxxx")
        channel_id = "xxxxxxx"

        #Lähetetään viesti slack-kanavaan, jos se ei onnistu niin sitten telegram-viesti ja mikäli telegram viestinkin lähetys epäonnistuu lähetetään sähköpostiviesti
        try:
            client.chat_postMessage(
                channel= channel_id,
                text= "Palvelin: %s \nViesti: %s" %(self.where, message)
            )
        except SlackApiError as e:
            try:
                r = requests.post(url = ("https://api.telegram.org/xxxxxx","xxxxxxx",("Palvelin: %s \n%s" %(self.where,message))))
            except:
                try:
                    requests.post(url = ("https://api.houston.tampuuri.fi/SendEmailxxxxxxx","xxxxxxx",("Palvelin: %s \n%s" %(self.where,message))))
                except:
                    print("Nothing works!!")

            print(f"Error: {e}")
```

Kuvio 16. Viestinlähetysmoduuli

Viestinlähetysmoduuli saa argumentteina kutsussa palvelimen nimen sekä tiedon missä osassa virhe on esiintynyt. Moduuli muodostaa sille välitetyillä tiedoilla lähetettävän virheviestin. Viestin muodostamisen jälkeen viestiä yritetään ensin lähettää Slackin rajapinnan kautta. Jos lähetys epäonnistuu, yritetään lähettää virheviesti Telegramin rajapinnan kautta ja jos tämäkin lähetystapa epäonnistuu, on yrityksellä käytössä oma sisäinen rajapinta, minkä kautta voi lähettää sähköpostiviestejä.

Kaikilla näillä viestinlähetysmetodeilla on erilaiset rajapintakutsut. Slack viestinlähetystä varten tarvitaan slack workspace, slack app sekä kanava mihin viestit lähetetään. Slack appille voidaan luoda botti, minkä kautta viestejä voidaan sitten välittää workspacessa olevaan kanavaan. Tässä työssä käytettiin Slackin rajapintaa valmiin Slack sdk moduulin kautta. Slack sdk moduuli haetaan tiedoston alussa, jonka jälkeen sitä voidaan käyttää myöhemmin koodissa.

Telegramissa riittää, että käyttäjä on rekisteröitynyt. Kun käyttäjä on rekisteröitynyt, hän voi luoda botin, jonka voi lisätä kanavalle, mihin viestejä halutaan lähettää. Rajapintakutsua tehdessä annetaan kutsun mukana botin nimi sekä botin valtuutusavain.

Yrityksen oma rajapinta on saatavilla ainoastaan sisäverkosta ja siihen on ennalta määritettyjä valtuutusavaimia. Rajapinnasta löytyy myös mahdollisuus lähettää tekstiviestejä.

8 Jatkokehitysehdotukset

Jatkokehityksenä toteutetaan valvontajärjestelmän jäljelle jäävät osat, eli verkko- ja sovelluskerroksen valvontaohjelmat. Valvontajärjestelmälle kehitetään myös graafinen käyttöliittymä, jotta virheiden seurantaa saadaan visuaalisemmaksi. Käyttöliittymässä tulee olemaan jonkinlainen listanäkymä virheistä. Listanäkymässä virheille olisi määrätty vakavuusluku ja virheet listattaisiin niin, että vakavimmat virheet ovat ylimpänä ja eri värillä huomion herättämistä varten. Virheiden korjausta voitaisiin automatisoida, esimerkiksi jos sivusto ei vastaa, voitaisiin ensiapuna sivusto käynnistää uudelleen automaattisesti. Jokaisesta ohjelman automaattisesti tekemästä toimenpiteestä lähetetään viesti samaan paikkaan, kun mihin alkuperäinen virheviestikin on lähetetty.

Virheille kehitetään myös vakavuus- tai ilmaantuvuusluvun määritysalgoritmi. Tämä helpottaa jatkossa virheiden esittämistä käyttöliittymässä halutussa järjestyksessä.

Ohjelman kehityksen aikana huomasin, että palvelimen etähallintapaneelista löytyy suoraan mahdollisuus käyttää syslog -edelleen lähetystä tai snmptrap -ominaisuuksia. Lyhyellä tutustumisella näihin, vaikuttavat ne huomattavasti paremmilta ratkaisuilta valvontaongelmaan kuin tässä työssä kehitetty valvontaohjelma. Työssä kehitetyn mukainen valvontaohjelma olisi hyvä, mikäli palvelininfra koostuisi useiden eri valmistajien palvelimista, koska tällaisella ohjelmalla voitaisiin standardoida välitettävät viestit, että virheilmoitus olisi lukijalle aina samanlainen.

Palvelimen välittämä syslog- tai snmptrap -tieto voitaisiin kerätä esimerkiksi Graylog-sovellukseen tai vastaavaan sovellukseen. Ilmainen versio Graylogista tukee viestien lähetystä esimerkiksi webhookien kautta ja maksullisessa versiossa voi määritellä suorittamaan tietyn ohjelman, kun se havaitsee ennalta määritellyn virheen. Mahdollisuus suorittaa ohjelmia tietyn virheen sattuessa voisi tarkoittaa mahdollisuutta automatisoida virheenkorjausta.

Syslogin ja snmptrapin edut verrattuna toteutettuun ohjelmaan ovat ne, että palvelin välittää tiedon heti kun ongelma havaitaan. Ohjelma taas kyselee rajapinnan kautta tietoja 30 minuutin välein, eli tiedon saamisessa on pahimmillaan 30 minuutin viive.

Huonoina puolina voisi pitää sitä, että jos palvelin ei saa virtaa tai sen etähallintaverkkosovittimen verkkoyhteys on katkennut, ei se pysty myöskään lähettämään siitä viestiä. Toiselta palvelimelta taas rajapinnan kautta kyseltäessä tästä saataisiin virhe, joka voitaisiin välittää eteenpäin.

Todennäköisesti tulevaisuudessa yritykselle suunnitellaan valvontaohjelma mikä on näiden kahden yhdistelmä.

9 Oman oppimisen arviointi

Tämän työn ansiosta pääsin tutustumaan useisiin erilaisiin rajapintoihin ja niiden toimintaan. Olin aikaisemminkin käyttänyt rajapintoja, mutta en ollut tutustunut niiden toiminnallisuuksiin näin kattavasti, kun tämän työn kannalta oli tarpeellista. Pääsin myös tutustumaan syvällisesti verkkopalvelun palvelininfraan sekä verkkoratkaisuihin. Ohjelmointikielien osalta ei työn aikana juurikaan tullut uutta, vaan pääosin hyödynsin aikaisemmin oppimaani.

Opinnäytetyön raportin osalta huomasin, että sen kirjoittaminen ei ollutkaan niin helppoa kuin olin olettanut. Luulin opinnäytetyötä aloittaessa, että ohjelman tekeminen olisi vaikeampi, mutta kun aloin kirjoittamaan raporttia tajusin, että se olikin vaikeampaa kuin ohjelman suunnittelu, määrittely, toteuttaminen ja asentaminen yhteensä. Erityisen vaikeaa tuntui olevan löytää luotettavia lähteitä teoriaosuuteen, koska ohjelmoida voi niin monella eri tavalla ja tietoa on saatavilla hyvin kattavasti internetistä, niin piti aina pystyä varmistamaan, että lähde on sellaisen henkilön kirjoittama, että sen kirjoittajan saavutukset voidaan tarkastaa.

Lähteet

Painetut

Kasurinen j. 2013. Ohjelmistotestauksen käsikirja. Docendo.

Dooley J. 2011. Software Development and Professional Practice. Springer.

Sähköiset

Visma Tampuuri. 2021. Tietoa meistä. Yrityksen verkkosivu. Viitatttu 6.5.2022. https://www.tampuuri.fi/visma-tampuuri/tietoa-meista/

Visma Tampuuri. 2022. Visma on ostanut Hausvise – isännöintijärjestelmää kehittävän Festum Software Oy:n. Viitattu 6.5.2022. https://www.tampuuri.fi/2022/04/visma-on-ostanut-hausvise-isannointijarjestelmaa-kehittavan-festum-software-oyn/

Kennedy P. 2018. Explaining the Baseboard Management Controller or BMC in Servers. Viitattu 6.5.2022. https://www.servethehome.com/explaining-the-baseboard-management-controller-or-bmc-in-servers/

IBM Cloud Education. 2020. Application Programming Interface (API). Viitattu 6.5.2022. https://www.ibm.com/cloud/learn/api

SendGrid Team. 2014. Webhook vs API: What's the Difference Between Them? Viitattu 6.5.2022. https://sendgrid.com/blog/webhook-vs-api-whats-difference

Python. What is Python? Executive Summary. Viitattu 6.5.2022. https://www.python.org/doc/essays/blurb/

Microsoft. 2022. What is PowerShell? Viitattu 6.5.2022. https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.2

Oracle. 2020. What is SaaS (Software as a Service)? Viitattu 6.5.2022. https://www.oracle.com/applications/what-is-saas/

Martin M. 2022. SDLC (Software Development Life Cycle): What is, Phases & Models. Viitattu 6.5.2022. https://www.guru99.com/software-development-life-cycle-tutorial.html

Hughey D. 2009. The Traditional Waterfall Approach. Viitattu 6.5.2022. https://www.umsl.edu/~hugheyd/is6840/waterfall.html

W3schools. 2022. SDLC Waterfall Model. Viitattu 6.5.2022. https://www.w3schools.in/sdlc/waterfall-model.

Martin M. 2022. What is Waterfall Model in SDLC? Advantages and Disadvantages. Viitattu 6.5.2022. https://www.guru99.com/what-is-sdlc-or-waterfall-model.html

Doshi K. 2019. Part 1 – Understanding Software Development Process. Viitattu 6.5.2022. https://www.browserstack.com/guide/learn-software-development-process

Agile Alliance. 2021. The Agile Manifesto. Viitattu 6.5.2022. https://www.agilealliance.org/agile101/the-agile-manifesto/

Agile Alliance. 2022. The 12 Principles behind the Agile Manifesto. Viitattu 6.5.2022. https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Yu B., Wooi K. & Wai Y. 2012. Software Development Life Cycle AGILE vs Traditional Approaches. Viitattu 6.5.2022. https://ku-fpg.github.io/files/agile-traditional.pdf

Diogo E. 2022. Top 5 main Agile methodologies: advantages and disadvantages. Viitattu 6.5.2022. https://www.xpand-it.com/blog/top-5-agile-methodologies/

Schwaber K. & Sutherland J. 2020. The 2020 Scrum Guide. Viitattu 6.5.2022. https://scrumguides.org/scrum-guide.html

Luukkainen M. 2021. Ohjelmistotuotanto 2021 Osa 2. Viitattu 6.5.2022. https://ohjelmistotuotanto-hy.github.io/osa2/

Aristek System. 2021. Software Development Life Cycle (SDLC) Overview. Viitattu 6.5.2022. https://aristeksystems.com/blog/sdlc-overview

Javatpoint. 2022. Software Development Life Cycle (SDLC). Viitattu 6.5.2022. https://www.javatpoint.com/software-development-life-cycle

Dell. 2022. iDRAC9 Redfish API. Viitattu 6.5.2022. https://developer.dell.com/apis/2978/versions/5.xx/docs/0WhatsNew.md

W3Schools. 2022. JSON – Introduction. Viitattu 9.5.2022. https://www.w3schools.com/js/js_json_intro.asp

Postman. 2022. About Postman. Viitattu 9.5.2022. https://www.postman.com/company/about-postman/

Kapersky. 2022. IP-osoite – määritelmä ja selitys. Viitattu 11.5.2022. https://www.kaspersky.fi/resource-center/definitions/what-is-an-ip-address

Pöyhönen I. 2005. Lääkintälaitteiden ohjelmistot. Suunnittelun kehityskohteita vesiputous- ja XP-mallin näkökulmasta. Viitattu 18.5.2022. https://www.vttresearch.com/sites/default/files/pdf/tiedotteet/2005/T2320.pdf

Kuviot